Oracle DBA实践操作指南

Oracle 11g R2 DBA操作指南

Learn Oracle from Oracle Certified Master

林树泽　卢芬　编著



完整、详尽、真实的Oracle数据库实用指导

- 全面讲解Oracle 11g R2数据库操作
- 通过示例说明概念和基本操作，方便读者把握相应的内容
- 内容体系涵盖Oracle DBA应该掌握的各方面知识
- 结合笔者Oracle数据库管理和维护经验，使读者入门更容易
- 可作为Oracle 11g R2数据库管理操作手册
- 一册在手，管理无忧

清华大学出版社

Oracle 11g R2
DBA操作指南

林树泽　卢芬　编著



清华大学出版社
北　京

内 容 简 介

Oracle 数据库是一款优秀且应用广泛的关系数据库管理系统。本书全面、详细地讲解了 Oracle 11g R2 数据库管理技术，是学习 Oracle 数据库管理的实用教材和参考书。

本书共分 26 章，通过几百个范例详尽讲解了 Oracle 11g R2 数据库安装与卸载、SQL 语言、各种数据库对象、数据库备份与恢复、用户与系统管理、企业管理器（EM），以及数据库性能优化等技术。书中每章的内容不仅概念清晰、操作步骤明了、示例丰富，而且更侧重于满足实际工作的需要。

本书适合 Oracle 数据库系统管理的初学者以及 Oracle 11g R2 DBA 参考。书中的内容覆盖了 OCA 考试的所有知识点，同样适合参加 OCA 考试的读者。

图书在版编目（CIP）数据

Oracle 11g R2 DBA 操作指南/林树泽，卢芬编著. —北京：清华大学出版社，2013（2018.5 重印）
ISBN 978-7-302-33910-6

I. ①O… II. ①林… III. ①关系数据库系统—指南 IV. ①TP311.138-62

中国版本图书馆 CIP 数据核字（2013）第 220416 号

责任编辑：夏非彼
封面设计：王　翔
责任校对：闫秀华
责任印制：李红英

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京中献拓方科技发展有限公司
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**36.75　　**字　　数：**940 千字

版　　次：2013 年 10 月第 1 版　　**印　　次：**2018 年 5 月第 6 次印刷
印　　数：7001~7500
定　　价：85.00 元

产品编号：046901-01

前　言

本书是一本介绍如何学习 Oracle 11g R2 数据库的入门书籍。Oracle 数据库已经成为当今市场的主流数据库产品。目前 Oracle 家族中已经不仅仅限于数据库这一产品，还包括操作系统、中间件等，Oracle 已经收购了 MySQL、SUN 等，其市场份额远远超过其他任何数据库产品，国内几乎所有大型企业，以及政府部门、军方都在使用 Oracle 数据库。

此领域市场职位缺口巨大，尤其是中高端人才更是紧缺。众多初学者都想掌握 Oracle 数据库技术，并迈入 Oracle 高端人才行列，这就产生了对入门技术的庞大需求，而 Oracle 也顺应此潮流推出了 OCA 认证体制，本书覆盖 OCA 考试的绝大部分知识点，也是初学者必须掌握的基础知识。本书结合笔者多年的 Oracle 数据库学习和维护经验，希望它可以成为 Oracle 数据库初学者的必备书籍，也希望准备 OCA 考试的人员把它作为一本很好的参考教材，同时书中对于知识点的介绍更多的是站在系统的全局角度考虑，读者可以从中获得全新的认识和实践体验。

本书内容

本书是一本全面讲述 Oracle 11g R2 数据库系统管理的图书，之所以选择 Oracle 11g R2 这个版本是因为 Oracle 已经不再提供对 Oracle 10g 的技术支持，Oracle 11g R2 版本是其目前支持的版本，这个版本的技术支持延续到 2017 年。本书开篇针对初学者非常详细地介绍了如何安装和配置 Oracle 数据库，这样读者能快速配置一个学习环境。通过本书的学习，读者能够全面地掌握 Oracle 数据库的体系架构，以及数据库管理、数据库安全、数据库优化和高可用等 DBA 所需要掌握的知识和技能。通过本书的学习和实践，相信读者能够完成基本的数据库管理工作，成为一名合格的 Oracle 数据库初级管理员。

全书共分 26 章，包括 Oracle 数据库基础、数据库启动与关闭、Oracle 数据库体系结构、SQL 语言概述、数据字典、网络连接管理、内存管理、用户管理和资源文件、管理控制文件、重做日志管理、管理归档日志、表空间和数据文件、UNDO 表空间管理、事务管理、角色管理、表管理、索引管理、系统和对象权限管理、视图、序列号和同义词、RMAN 备份与恢复数据库、闪回技术、手工管理的备份恢复、企业管理器（EM）、数据库优化等内容。

本书特点

- 知识体系涵盖 Oracle DBA 管理应掌握的各方面知识，覆盖了跨平台下 Oracle DBA 管理应具备的各方面知识和技能。
- 内容全面而深入，实例得当，切中 DBA 关注要点，描述内容由浅而深，详尽细致，能够帮助读者较好地掌握 DBA 知识和技能。
- 注重实践和应用，从数据库管理、开发、安全、性能优化和高可用等数据库管理员最关注的方面做了详尽的描述，使读者在实际应用时能够快速上手，并且在遇到问题时能够

在本书中获得学习参考。可谓一册在手，管理无忧。

- 实例详尽、图文并茂、示例清晰、所有案例均在实践环境中经过检验。

本书可以作为 Oracle 11g R2 数据库操作手册，可供 Oracle 数据库管理员、Oracle 数据库应用开发人员、Oracle 数据仓库工程师使用，还可以作为 Oracle 技术支持和培训机构、高等院校数据库课程的参考教材。

参与本书写作的除了林树泽、卢芬外，还有历铁帅、何会军、李渊、陈玉等人，他们为本书的创作做了大量的工作，在此表示衷心的感谢。

编　者

2013 年 8 月

目　录

第 1 章

Oracle数据库基础

这一章我们介绍数据库基础知识，这些知识包含了数据模型、关系数据库的历史、Oracle 数据库的发展简史、与数据库密切相关的 SQL 语言，以及计算机结构基础知识。读者只需要了解本章的内容即可。

1.1　什么是关系数据库

关系数据库是建立在关系数据模型基础上的数据库管理系统，在 20 世纪 70 年代曾经出现过不同的数据模型，如网络模型、层次模型，这两种数据模型都早于关系模型。网络模型和层次模型都使用指针作为数据连接处理用户的查询需求，数据组织非常复杂，无论从设计到维护都极为复杂，不利于推广使用。而关系数据库管理系统使用数据之间的关系来满足用户的复杂查询，数据库的设计更加简单，关系数据模型相对较早的两种数据模型具有更多的优势。下面我们重点理解什么是关系数据库模型。

1.1.1　关系数据库模型

关系数据库模型有 3 个核心的概念，即关系、属性和域。其中关系就是由行列组成的表。属性就是关系中的列。域是列取值的集合。下面以一个描述员工实体的表 EMP 来具体解释关系数据库模型内容，如图 1-1 所示。

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-12月-80	800		20
7499	ALLEN	SALESMAN	7698	20-2月 -81	1600	300	30
7521	WARD	SALESMAN	7698	22-2月 -81	1250	500	30
7566	JONES	MANAGER	7839	02-4月 -81	2975		20
7654	MARTIN	SALESMAN	7698	28-9月 -81	1250	1400	30

图 1-1　描述员工实体的表

表是关系模型的基本数据结构，所谓关系就是数据库中的表。表本身对应现实中的一个实体，如描述员工的表，员工就是关系数据模型要描述的实体。这里表 emp 中列表示实体的属性，如员工的 ID、名称、岗位、薪水以及入职时间等等。通过这个属性的集合来描述实体。而每一行成为一个元组，它表示一个具体的员工实体，如第一行即第一个元组为名称是 SMITH 的员工。所以员工表的每个元组描述了每一个具体员工的信息。

对于基本关系根据关系数据模型的要求必须满足基本的规则。要求一个表（关系）中的列的顺序是随意的，没有先后之分。表中每个元组的属性不能完全相同，即不能出现两个完全相同的员工，员工不能在一个表中重复输入。每一个具体的元组对应的属性将包含一个值。

对于表中的每个元组，通过一个成为键的属性来唯一标识，键可以由一个或者多个属性的组合来定义，目的是通过键可以唯一确定一个元组（一个具体的员工）。键是建立索引的依据，通过索引可以改善检索表中数据的速度。

1.1.2 关系数据模型的创始人

这一小节，我们认识一下关系数据模型的创始人科德（如图 1-2 所示），以纪念这位为数据库发展做出杰出贡献的计算机科学家。他也是图灵奖获得者。

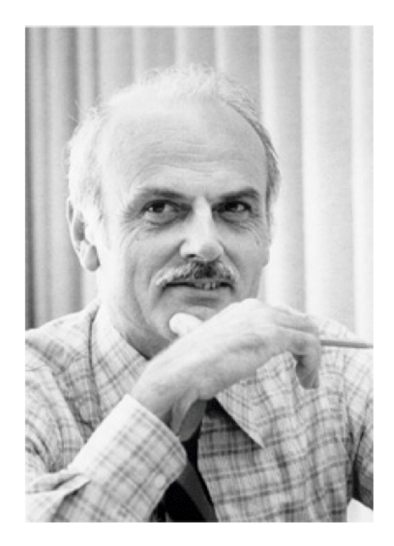


图 1-2 Edgar Frank Codd

埃德加·弗兰克·科德（Edgar Frank Codd，1923－2003）是密执安大学哲学博士，IBM 公司研究员，被誉为“关系数据库之父”，并因为在数据库管理系统的理论和实践方面的杰出贡献于 1981 年获图灵奖。1970 年，科德发表题为《大型共享数据库的关系模型》的论文，文中首次提出了数据库的关系模型。由于关系模型简单明了，具有坚实的数学理论基础，所以一经推出就受到了学术界和产业界的高度重视和广泛响应，并很快成为数据库市场的主流。20 世纪 80 年代以来，计算机厂商推出的数据库管理系统几乎都支持关系模型，数据库领域当前的研究工作大都以关系模型为基础。

科德考特原是英国人，1923 年 8 月 19 日生于英格兰中部的港口城市波特兰。第二次世界大战爆发以后，年轻的科德应征入伍在皇家空军服役，1942－1945 年期间任机长，参与了许多重大空战，为反法西斯战争立下了汗马功劳。二战结束以后，科德上牛津大学学习数学，于 1948 年取得学士学位以后到美国谋求发展。他先后在美国和加拿大工作，参加了 IBM 第一台科学计算机 701 以及第一台大型晶体管计算机 STRETCH 的逻辑设计，主持了第一个有多道程序设计能力的操作系

统的开发。他自觉硬件知识缺乏，于是在 60 年代初，到密歇根大学进修计算机与通信专业（当时他已年近 40），并于 1963 年获得硕士学位，1965 年取得博士学位。这使他的理论基础更加扎实，专业知识更加丰富。加上他在此之前十几年实践经验的积累，终于在 1970 年迸发出智慧的闪光，为数据库技术开辟了一个新时代。

1970 年以后，考特继续致力于完善与发展关系理论。1972 年，他提出了关系代数和关系演算的概念，定义了关系的并、交、投影、选择、连接等各种基本运算，为日后成为标准的结构化查询语言（SQL）奠定了基础。

1.2 Oracle数据库发展简史

本节我们了解一些 Oracle 数据库公司的一些历史，以及 Oracle 数据库的发展简要时间列表。通过对公司发展之初的理解或许可以给我们一些启示，而 Oracle 数据库发展时间列表展示了 Oracle 数据库不断满足市场需求，应用新技术的发展历程。

1.2.1 公司之初

1970 年的 6 月，IBM 公司的研究员科德在 Communications of ACM 上发表了那篇著名的《大型共享数据库的关系模型》（A Relational Model of Data for Large Shared Data Banks）的论文。从这篇论文开始，拉开了关系型数据库软件革命的序幕。

虽然早在 1970 年就诞生了关系模型理论，但是市场上迟迟不见关系型数据库管理软件的推出。主要原因是很多反对者认为关系型数据库速度太慢，比不上当时的层次式数据库。值得一笑的是，IBM 虽然 1973 年就启动了 System R 的项目来研究关系型数据库的实际可行性，也没有及时推出这样的产品，因为当时 IBM 的的 IMS（著名的层次型数据库）市场不错，如果推出关系型数据库，牵涉到 IBM 很多人的自身利益。再者，IBM 庞大复杂的官僚机构在决策上不那么灵活。

当拉里·埃里森偶然看到有关关系型数据库工作原型的一段描述时，他发现了一个其他公司错失的机遇。当时，尚未有企业致力于商业化这一技术。埃利森和他的共同创始人 Bob Miner 和 Ed Oates 认识到在关系型数据库模型方面存在极其巨大的商业潜力。

1977 年 6 月，Larry Ellison、Bob Miner 和 Ed Oates 在硅谷共同创办了一家名为软件开发实验室（Software Development Laboratories，SDL）的计算机公司（Oracle 公司的前身）。很快他们就弄出来一个不太像样的产品，或者说更像是一个 Demo。根据 Ellison 和 Miner 在前一家公司从事的一个由中央情报局投资的项目代码，他们把这个产品命名为 Oracle。因为他们相信，Oracle（字典里的解释有“神谕、预言”之意）是一切智慧的源泉。1979 年，SDL 更名为关系软件有限公司（Relational Software，Inc.，RSI），毕竟“软件开发实验室”不太像一个大公司的名字。1983 年，为了突出公司的核心产品，RSI 再次更名为 Oracle。

1.2.2 Oracle 数据库的发展历程

- 1977 年：1977 年 Larry Ellison、Bob Miner 和 Ed Oates 3 人创建了“软件开发实验室”，而后该转变为关系软件公司（RSI）。在 1983 年，RSI 公司更名为 Oracle 系统公司，后

来才是现在的 Oracle 公司。

- 1979 年：Oracle 数据库系统是第一个应用于商业的关系数据库系统（RDBMS）。在 1979 年，RSI 将 Oracle V2 作为第一个基于 SQL 的商务关系数据库系统，这成为关系数据库历史上的里程碑事件。
- 1983 年：Oracle 数据库的便携版本。Oracle V3 在 1983 年发行，是第一个运行在大型机、微机和 PC 上的商务关系数据库系统。该关系数据库系统使用 C 语言编写，使得它对多个系统平台提供了应用接口。
- 1985 年：在并发控制、数据分布以及可扩展性方面的提升。V4 引进了多版本读一致性。V5 在 1985 年发行，它支持客户端/服务器计算和分布式数据库系统。V6 在磁盘 I/O，行锁、可扩展性以及备份恢复方面有较大提升。V6 引入了 PL/SQL 语言的第一个版本，它是对 SQL 的私有过程的扩展。
- 1992 年：PL/SQL 存储程序的单元。Oracle 7 在 1992 年发行，引入了 PL/SQL 存储过程和触发器。对象和分区 Oracle 8 在 1997 年发行，作为第一个面向对象的数据库，支持许多新的数据类型。除此之外，Oracle 支持大表的分区功能。
- 1999 年：网络计算。Oracle 8i 数据库在 1999 年发行，它提供了网络协议的原生支持和服务器端对 Java 的支持。Oracle8i 为网络计算而设计，使得数据库可以部署到多层环境中。
- 2001 年：RAC。Oracle 9i 数据库在 2001 年发行，它引入了 RAC，使得多个实例可以同时访问单一的数据库。另外，Oracle XML 数据库可以存储和查询 XML。
- 2003 年：网格计算。Oracle 10g 在 2003 年发行，它引入了 Grid Computing。这个版本通过廉价的服务器创建的网格基础设施实现计算资源的虚拟化。最终目的是使得数据库可以自管理和自调优。Oracle ASM 通过虚拟化和简单化数据库存储管理来达到子管理和自调优的目的。
- 2007 年：可管理性、可诊断性和可获得性。Oracle 11g 在 2007 年发行，引入了许多新特性，使管理员和开发人员快速适应变化的商业环境需求。其核心是简化信息基础架构，通过 consolidating 信息和在任何可能的地方使用自动操作。

1.3 数据库RDBMS

关系数据库管理系统（Relational DataBase Management System，RDBMS），简单地说，它是用户和数据的接口，它处理、协调用户对数据库的各种请求。

在关系数据库理论的创始人 E.F.Codd 博士发表的论文《大规模共享数据银行的关系型模型》中提及 RDBMS，随后在这篇论文的基础上设计开发了 RDBMS 系统，第一个 RDBMS 系统是 IBM 的 System R 关系数据库系统。经过三十多年的发展，RDBMS 已经发展的十分成熟，目前业界流行的关系数据库管理系统产品包括 Oracle、DB2、MySQL 以及 SQL Server 等。下面我们简单列出 RDBMS 的基本功能。关系数据库管理系统如图 1-3 所示。

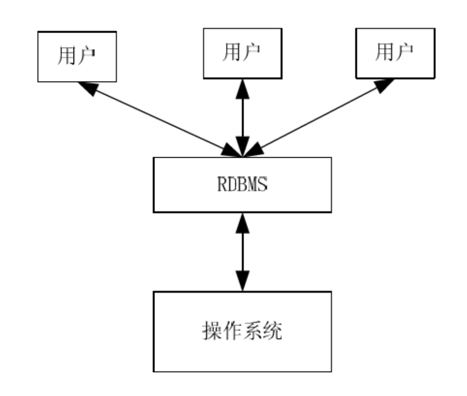


图 1-3　关系数据库管理系统

数据库管理系统是位于用户与操作系统之间的一层数据管理软件，用于科学地组织和存储数据，高效地获取和维护数据。图 1-4 是 DBMS 的基本功能。

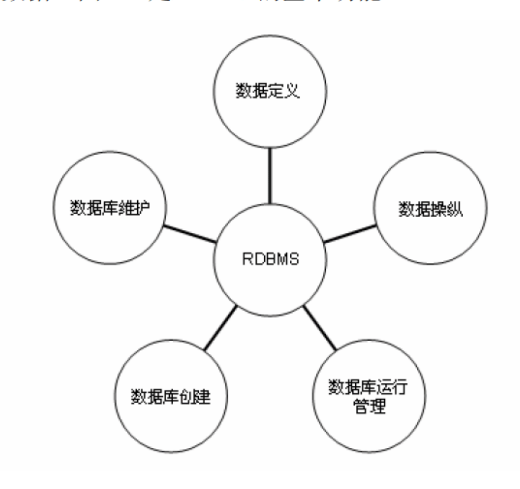


图 1-4　关系数据库管理系统的基本功能

DBMS 的主要功能包括数据定义功能、数据操纵、数据库的运行管理、数据库的建立和维护功能。解析 DBMS 是一个大型的复杂的软件系统，是计算机中的基础软件。

1.4　SQL语言简介

操作数据库的语言称为结构化查询语言（Structure Query Language，SQL）。是直接操作数据库的语言，如最常使用的 DML 语句。下面我们进一步介绍 SQL 语言。更详细的内容将在本书的第 5 章介绍。

1.4.1　SQL 语言概述

SQL 是一种用于数据库访问的非过程化语言。用户通过 SQL 描述其目标，之后 SQL 语言

编译器自动地生成执行过程，控制数据库执行用户所期望的操作。

Oracle SQL 包括许多对 ANSI/ISO 标准 SQL 语言的扩展，Oracle 工具及应用程序也增加了额外的语句。用户可以使用 Oracle 工具 SQL*Plus 或 Oracle 企业管理器（Oracle Enterprise Manager）对 Oracle 数据库执行任意的 ANSI/ISO 标准 SQL 语句，以及这些工具提供的额外的语句或函数。

所有的数据库操作都是通过 SQL 提交给数据库的，但 Oracle 工具及应用程序能够简化或隐藏实际的 SQL。SQL 之外的任何数据访问方式均会绕过 Oracle 内置的安全特性，有可能对数据的安全性及完整性造成破坏。

1.4.2 SQL 语句的操作

对 Oracle 数据库内存储的信息所执行的所有操作都是通过 SQL 语句（statement）执行的。在数据库中 SQL 语句操作的对象绝大多数都是表或者视图对象。一条 SQL 语句相当于一条计算机程序或指令。因此 SQL 语句必须包含一段完整的 SQL 语法，例如：

```
Select id , name, age from student;
```

该语句的含义是从表 student 中查询列 id、name、age 记录，其中 select、from 是关键字。

Oracle SQL 语句可以分为以下几类：

- 数据操作语言语句
- 数据定义语言语句
- 事务控制语句
- 会话控制语句
- 系统控制语句
- 嵌入 SQL 语句

1. 数据操作语言语句

数据操作语言（Data Manipulation Language，DML）语句的作用是查询或操作已有方案对象内的数据。用户利用 DML 语句可以完成以下工作：

- 从一个或多个表和视图中查询数据（SELECT），获取（fetch）操作是可滚动的（scrollable）（见“可滚动游标”）
- 向表或视图中加入新数据行（INSERT）
- 修改表或视图中已有数据行的列值（UPDATE）
- 根据判断条件为表及视图插入或更新数据行（MERGE）
- 从表或视图中删除数据行（DELETE）
- 查询 SQL 语句的执行计划（EXPLAIN PLAN）
- 对表或视图加锁（lock），临时地限制其他用户访问此对象（LOCK TABLE）

2. 数据定义语言语句

数据定义语言（Data Definition Language，DDL）语句的作用是定义或修改方案对象（schema

object）的结构，以及移除方案对象。用户利用 DDL 语句可以完成以下工作：

- 创建、修改、移除方案对象及其他数据库结构，包括数据库自身及数据库用户（CREATE，ALTER，DROP）
- 修改方案对象名称（RENAME）
- 删除方案对象的所有数据，但不移除对象结构（TRUNCATE）
- 授予或收回权限及角色（GRANT，REVOKE）
- 打开或关闭审计选项（AUDIT，NOAUDIT）
- 向数据字典中添加注释（COMMENT）

3. 事务控制语句

事务控制语句（transaction control statement）的作用是管理 DML 语句对数据的修改，以及将逻辑上相关的 DML 语句组织为事务。用户利用事务控制语句可以完成以下工作：

- 将事务对数据的修改永久地保存到数据库（COMMIT）
- 还原事务对数据的修改，可还原到事务开始处或任意保存点（savepoint）（ROLLBACK）
- 设置保存点以标识回滚位置（SAVEPOINT）
- 设置事务的属性（SET TRANSACTION）

4. 会话控制语句

会话控制语句（session control statement）用于管理用户会话的属性。用户利用会话控制语句可以完成以下工作：

- 执行特定操作，修改当前会话，例如启用或禁用 SQL 跟踪功能（SQL trace facility）（ALTER SESSION）
- 为当前会话启用或禁用角色（role），即一组权限的集合（SET ROLE）

5. 系统控制语句

系统控制语句(system control statement)用于修改 Oracle 数据库实例的属性。ALTER SYSTEM 是唯一的系统控制语句。用户可以使用此语句修改实例设置（例如共享服务进程的最小数量）、终止进程（kill session）或执行其他操作。

6. 嵌入 SQL 语句

用户可以使用嵌入 SQL 语句（embedded SQL statement）将 DDL、DML 及事务控制语句加入到以过程化语言编写的程序中。Oracle 预编译器（precompiler）能够处理这样的代码。用户利用嵌入 SQL 语句可以完成以下工作：

- 定义、分配及释放游标（cursor）（DECLARE CURSOR，OPEN，CLOSE）
- 选择一个 Oracle 数据库并进行连接（DECLARE DATABASE，CONNECT）
- 分配变量名（DECLARE STATEMENT）
- 初始化描述符（descriptor）（DESCRIBE）
- 设定如何处理错误及警告（WHENEVER）
- 解析并执行 SQL 语句（PREPARE，EXECUTE，EXECUTE IMMEDIATE）

- 从数据库中取回数据（FETCH）

1.5 本章小结

本章介绍了数据库的基础知识，以帮助读者了解数据模型、关系数据库发展中的重要人物以及 Oracle 数据库的发展历史简要。在数据模型中我们介绍了层次模型、网状模型以及关系模型。Edgar Frank Codd 是关系数据库的创始人，他在该领域做出了一系列的重要贡献，回望这些，使得我们这些初学者不敢懈怠而勤奋求知。最后我们介绍了操作关系数据库的 SQL 语言，这是数据库操作的核心，所有的关系数据库操作最终要转化为 SQL 语句的操作，我们介绍了 SQL 语句的几种类型，并对其作用做了简要罗列，读者在后续的学习中可逐步体会这部分内容。

第 2 章

◂ Oracle 11g数据库初体验 ▸

学习 Oracle 数据库需要理论与实践并重，边看书边实践是最快的学习方法，所以在学习之前读者首先需要学会安装 Oracle 11g 数据库软件并创建数据库，学会使用基本的数据库管理工具访问数据库。在 Oracle 的不同数据版本中，数据库软件对于计算机硬件的要求不同，基本趋势是版本越高对计算机硬件要求越高，数据库管理越趋于自动化，主要体现在对内存和 CPU 的要求。本章我们介绍安装数据库软件的操作系统需求以及安装过程，演示在 Windows 系统和 Linux 系统上安装 Oracle 11g 数据库的系统需求以及创建过程。最后介绍如何删除数据库软件。

2.1　安装数据库的环境要求

数据库软件是运行在操作系统上的，它毕竟要消耗操作系统的各种资源如内存、CPU 以及 I/O 等，所以在安装 Oracle 数据库软件之前最好阅读相关随机文档，获得该软件对于操作系统的要求。然后再安装数据库软件。

Oracle 数据库软件可以安装在 Windows、Linux、HP-UNIX、AIX 等不同操作系统平台上。同样对于各自的平台，不同版本的 Oracle 数据库软件对于系统硬件要求不同，下面是 Oracle 11g Release 11.2.0.1.0 数据库在 WINDOWS 32-Bit 系统上对于硬件的需求：

- 物理内存（RAM）最小为 1GB
- 虚拟内存为物理内存的 2 倍
- 高级安装的磁盘空间为 5.15GB
- Intel 兼容处理器
- Video adapter 256 colors
- 显示器分辨率最小为 1024 × 768

笔者的电脑是 2GB 内存，操作系统为 Windows XP，要安装 Oracle 11g Release11.2.0.1.0 版本的数据库软件，下面我们给出具体的安装过程。

2.2　Windows环境下Oracle 11g的安装步骤

将下载的软件解压到一个目录下，默认在解压目录创建 database 文件夹，所有解压后的文件

都放在这个文件夹里。启动 Universal Installer，如图 2-1 所示。



图 2-1　启动 Universal Installer

然后会弹出对话框开始数据库软件的安装之旅，这个步骤是安装选项，可以省略，但是在安装生成数据库时，最好填写完整然后点单击下一步，如图 2-2 所示。



图 2-2　配置安全更新

接下来就是弹出如图 2-3 所示的对话框，提示有 3 个选项：创建和配置数据库、仅安装数据库软件以及升级现有数据库，显然，第一次安装数据库，我们选择前两项的一个，选择第一个即创建并配置数据库选项，这个选择的结果是整个安装过程包括两个部分，一个是安装数据库软件即 RDBMS，然后是通过 DBCA 来创建数据库。至于第二个选项是只安装数据库软件，然后再通过工具或者人工方式创建数据库，第二个选项我们在 Linux 环境下 Oracle 11g 的安装中再介绍。



图 2-3　安装选项

在选择创建并配置数据库之后，单击下一步按钮，如图 2-4 所示。要求选择是桌面类还是服务器类，其说明已经很详细了，这里我们选择桌面类。



图 2-4　选择系统类

然后单击下一步，如图 2-5 所示，这个是典型安装，要求我们填写相关目录、选择字符集以及数据库类型，并设置数据库口令，这个口令需要一定的复杂度，要求有字母带小写、数字不少于 8 个字符。在打开这个对话框时，其实除了管理口令外其他都是默认选项，对于 Oracle 基目录、软件位置和数据库文件位置，安装软件会自动探测一个满足硬盘大小需求的磁盘，自动填写以上目录。对于字符集默认是选择与操作系统一致的字符集。在典型安装这个步骤中，除了管理口令，我们都选择默认安装。

图 2-5 典型安装

在确认典型安装的内容后，单击下一步，就进入先决条件检查阶段，这个翻译有点拗口，其实就是检查当前操作系统的环境是否满足 Oracle11g 软件的安装需求，如果不满足会具体说明，此时需要重新对系统作出调整，以满足要求。如图 2-6 所示。



图 2-6 先决条件检查

经过检查，确实发现笔者的操作系统有不满足 Oracle 11g 安装需求的项目，即空闲空间不满足要求，如图 2-7 所示。



图 2-7　先决条件检查结果

在删除笔者 C 盘下部分文件后，释放部分空间，然后重新检查先决条件，发现没有不满足 Oracle 11g 安装的项目，则单击下一步继续安装。弹出如图 2-8 所示对话框，开始安装数据库软件。



图 2-8　安装数据库产品

其实，这个步骤包含几个过程，复制文件、安装程序文件和数据库配置，这个配置实际就是安装两个工具，一个是 NETCA，一个是 DBCA，如图 2-9 所示。



图 2-9　安装 Oracle 工具

第一个工具 NETCA（Oracle Net Configuration Assistant）用于配置网络、启动监听等，第二个工具 DBCA（Oracle Database Configuration Assistant）用于安装数据库，这两个工具安装后会自动后台运行，首先自动配置监听（在默认端口 1521），配置网络设置（NETCA 文件），然后启动 DBCA 建库，此时会弹出建库对话框，如图 2-10 所示。



图 2-10　使用 DBCA 建库

在数据库创建成功后，上述窗口退出，自动弹出如图 2-11 所示对话框，提示数据库成功安装，并清晰说明数据库信息、加密密钥文件位置、Database Control URL（EM 企业管理器）以及用户名和密码信息。

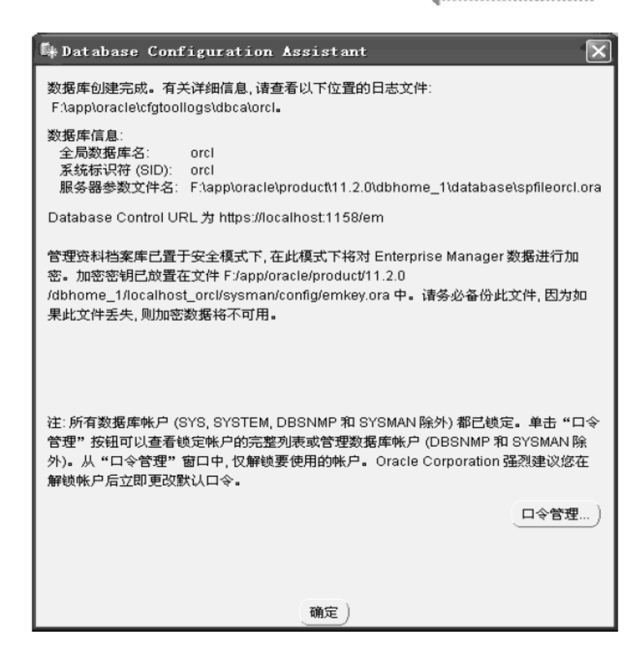


图 2-11　成功创建数据库信息

此时监听已经启动，监听器在通过网络访问数据库服务器时使用，客户端通过连接工具或者程序连接数据库，监听负责将用户连接请求交给数据库服务器进程。下面我们查看当前的监听器状态，如例子 2-1 所示。

例子 2-1　检查监听器状态

```
    D:\>lsnrctl status

    LSNRCTL for 32-bit Windows: Version 11.2.0.1.0 - Production on 03-12 月-2012
11:30:58

    Copyright (c) 1991, 2010, Oracle.  All rights reserved.

    正在连接到 (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC1521)))
    LISTENER 的 STATUS
    ------------------------
    别名                      LISTENER
    版本                          TNSLSNR for 32-bit Windows: Version 11.2.0.1.0 -
Production
    启动日期                  03-12 月-2012 10:14:48
    正常运行时间              0 天 1 小时 16 分 13 秒
    跟踪级别                  off
    安全性                    ON: Local OS Authentication
    SNMP                       OFF
    监听程序参数文件
    F:\app\oracle\product\11.2.0\dbhome_1\network\admin\listener.ora
    监听程序日志文件
    f:\app\oracle\diag\tnslsnr\oracle-linshuze\listener\alert\log.xml
    监听端点概要...
```

```
    (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(PIPENAME=\\.\pipe\EXTPROC1521ipc)))
    (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=127.0.0.1)(PORT=1521)))
服务摘要..
服务 "CLRExtProc" 包含 1 个实例。
  实例 "CLRExtProc", 状态 UNKNOWN, 包含此服务的 1 个处理程序...
服务 "orcl" 包含 1 个实例。
  实例 "orcl", 状态 READY, 包含此服务的 1 个处理程序...
服务 "orclXDB" 包含 1 个实例。
  实例 "orcl", 状态 READY, 包含此服务的 1 个处理程序...
命令执行成功
```

2.3 SQL Plus工具以及scott用户

既然成功安装了数据库，就需要维护和管理它，Oracle 提供了一个工具 SQL*Plus 来完成数据库的管理和维护任务，虽然 Oralce 较高版本的数据库软件中提供了大量的图形化管理手段，但是学会使用 SQL*Plus 工具仍然是 DBA 的一项重要基本功，因为有时图形化工具或许不能使用，而 SQL*Plus 则往往工作很好。

在 Oracle 11g 中可以如图 2-12 所示打开 SQL*Plus 工具。

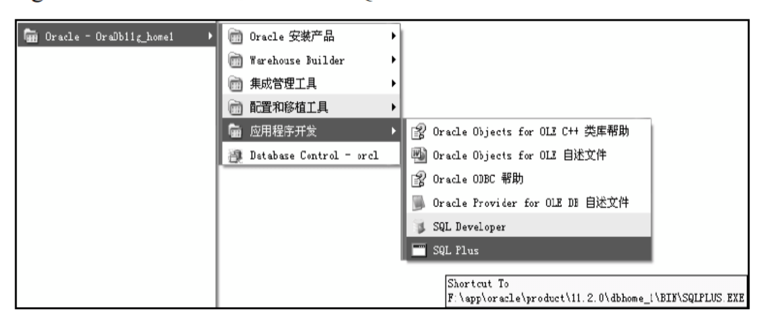


图 2-12　打开 SQL*Plus

单击图 2-12 中所示的“SQL Plus”选项，弹出如图 2-13 所示的对话框。此时提示“请输入用户名：”，在后面输入 scott，此时会继续提示“输入口令”，如图 2-14 所示，在口令中输入 tiger，因为是本机上安装的数据库可以不考虑“主机字符串操作”。在输入用户名和口令后，按“回车”键，执行结果如图 2-15 所示。



图 2-13　SQL*Plus 对话框

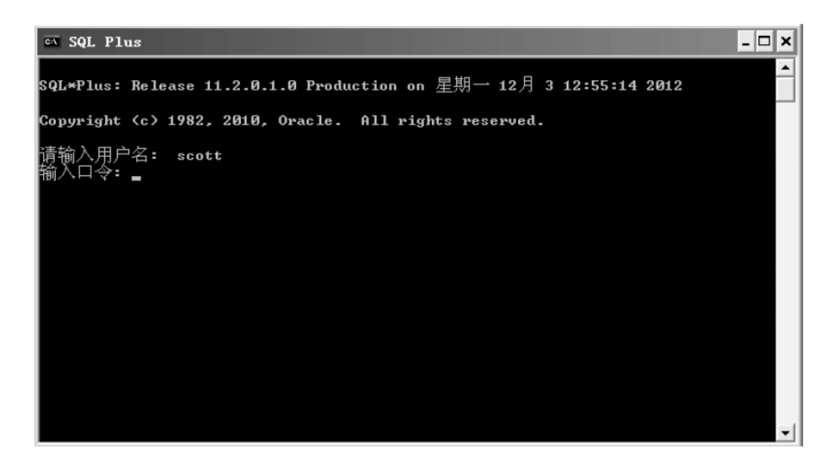


图 2-14　使用 SCOTT 用户登录数据库

在图 2-15 中提示错误（ERROR），错误信息是“the account is locked”，显然数据库告知用户 SCOTT 被锁定了。只有解锁后才可以使用该用户登录数据库。

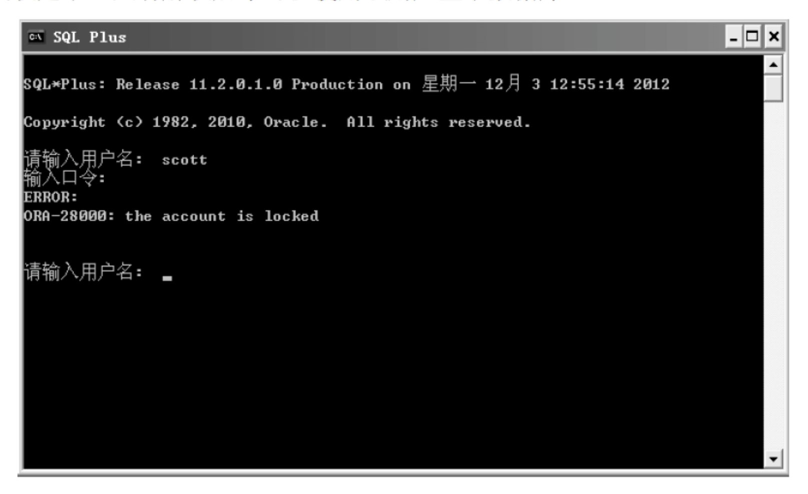


图 2-15　无法使用锁定的用户登录数据库

在图 2-11 中可以看到除了那 4 个用户外其他所有用户都已经锁定了，所以我们使用 SYSTEM 用户登录，密码为 Oracle1234，我们再次输入用户名和密码，如图 2-16 所示，数据连接成功，进入“SQL>”状态。此时，我们就可以使用 SQL*Plus 完成数据库的维护工作了，因为在以后的学习中，我们需要使用 SCOTT 用户来学习使用各种 SQL 语句，所以先解锁 SCOTT 用户，在解锁 SCOTT 用户时，使用数据库维护指令 alter user scott，如图 2-17 所示。

```
SQL Plus
SQL*Plus: Release 11.2.0.1.0 Production on 星期一 12月 3 12:55:14 2012

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

请输入用户名:  scott
输入口令:
ERROR:
ORA-28000: the account is locked

请输入用户名:  system
输入口令:

连接到:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> _
```

图 2-16　用 SYSTEM 用户登录数据库

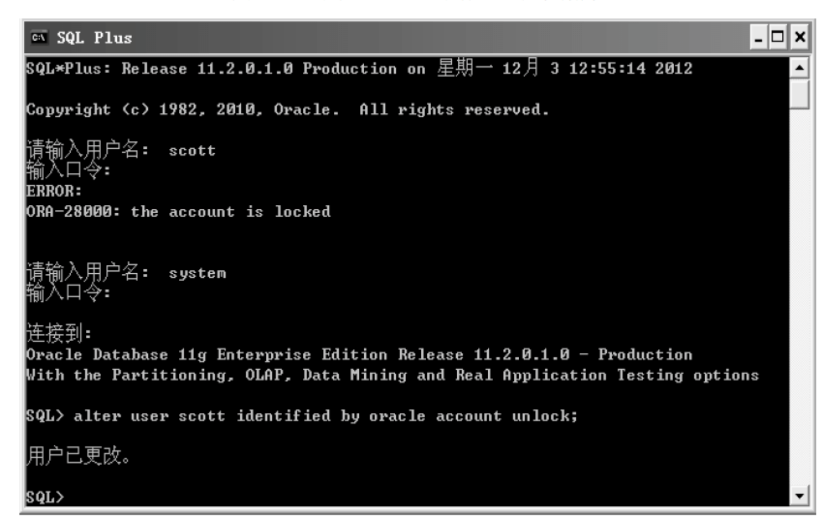


图 2-17 解锁 SCOTT 用户

图 2-17 所示，我们输入的指令执行成功，修改 SCOTT 用户密码为 oracle，并解锁账户。提示“用户已更改”，接下来就可以使用 SCOTT 用户了。

在安装数据库时，用户 SCOTT 是默认安装的而且密码默认为 tiger，其实 SCOTT 是 Oracle 数据库公司早期的一个程序员，而 tiger 则是他的一只宠物猫的名字，或许是为了记住这位早期优秀的数据库程序员而一直保留了这个用户，读者可以使用该用户下的几个表学习 Oracle 数据库 SQL 语句。

我们就是在 SQL*Plus 中输入各种类型的 SQL 语句或者执行某些.SQL 脚本（由多行 SQL 语句组成）来维护和管理数据库。如果读者在使用 SCOTT 用户学习时出现一些意外情况，如删除了数据无法恢复（没有学习备份前或没做任何备份表），或者该用户被他人删除等，可以使用一个脚本来恢复。该脚本文件位于 $ORACLE_HOME\RDBMS\ADMIN 下，脚本文件名为 scott.sql。$ORACLE_HOME 为本机安装 Oracle 数据库的主目录，即 F:\app\oracle\product\11.2.0\dbhome_1，

所以执行脚本文件的指令如下所示。

```
SQL> @ F:\app\oracle\product\11.2.0\dbhome_1\RDBMS\ADMIN\scott.sql
```

我们再提供一种启动 SQL*Plus 的方法，并使用解锁的 SCOTT 用户登录。具体步骤如下：

01 打开一个 DOS 窗口，如图 2-18 所示。

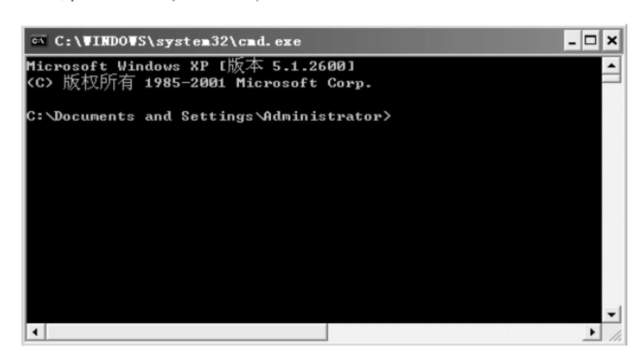


图 2-18 打开 DOS 窗口

02 在图 2-18 中输入“sqlplus /nolog”来启动 SQL*Plus 工具，如图 2-19 所示。

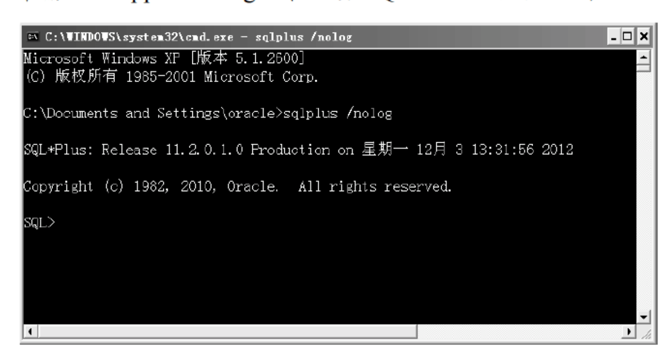


图 2-19 启动 SQL*Plus 工具

03 使用 CONNECT 指令使 SCOTT 用户登录数据库，如图 2-20 所示。

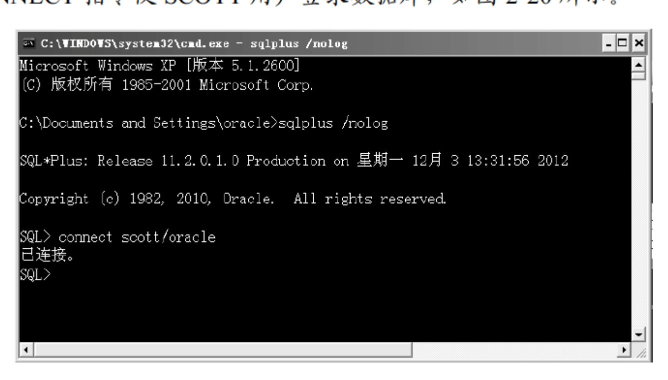


图 2-20 SCOTT 用户登录数据库

2.4 Linux环境下Oracle 11g的安装步骤

2.4.1 安装前的配置任务

在安装 Oracle 11g 数据库之前需要预配置任务，这些配置任务对操作系统内核、包、内存以及用户组和用户等有具体要求，下面我们按照具体步骤，一步步实现安装前的预配置任务。

1. 以 ROOT 用户登录

ROOT 用户具有最高的权限，使用 ROOT 用户登录系统。

2. 硬件需求

内存最少 1GB。内存大小与 SWAP 区设置有关。

（1）确定内存大小。

```
[root@localhost oracle]# grep MemTotal /proc/meminfo
MemTotal:      1034496 kB
```

当前机器是虚拟机，host 机器的内存是 2GB，所以分配了 1GB 内存给虚拟机用。

（2）内存与 SWAP 区查看其大小。

```
[root@localhost oracle]# free
           total       used       free     shared    buffers     cached
Mem:      1034496     285568     748928          0      17352     166988
-/+ buffers/cache:     101228     933268
Swap:     1020116          0    1020116
```

（3）11g 的 AMM 管理需要更多的共享内存。

在 11g 中如果设置了参数 memory_target 实现自动内存管理，此时设置的共享内存要大于参数 memory_target 或者 memory_max_target 参数设置的值，否则在启动时会导致 ORA-00845 错误。下面查看共享内存大小。

```
[root@localhost oracle]# df -h /dev/shm/
Filesystem            Size  Used Avail Use% Mounted on
none                 506M     0  506M   0% /dev/shm
```

（4）确定处理器架构与 Oracle 数据库软件相匹配。

```
[root@localhost oracle]# uname -m
i686
```

Linux x86 机器上对于其他磁盘空间的需求如下所示。

```
Installation Type Requirement for Software Files (GB)
Enterprise Edition 3.95
Standard Edition 3.88
Installation Type Disk Space for Data Files (GB)
Enterprise Edition 1.7
Standard Edition 1.5
```

安装软件所需空间和安装数据文件所需空间的最少值如上述所示。所以在初次安装 Oracle 11g 数据库作为学习库时，至少满足上述要求。有条件的话尽量使用更大的磁盘空间，毕竟当前的硬盘价格很廉价。

3. 软件需求

其实这个步骤可以在安装数据库软件时再进行，因为 Oracle Universal Installer 会自动测试软件环境是否满足要求。这里我们先完成各种软件配置，了解所需的具体配置内容。

（1）Oracle 11g R2 支持的操作系统。

```
On Linux x86:
- Asianux 2 Update 7
- Asianux 3
- Oracle Enterprise Linux 4 Update 7
- Oracle Enterprise Linux 5 Update 2
- Red Hat Enterprise Linux 4 Update 7
- Red Hat Enterprise Linux 5 Update 2
- SUSE Linux Enterprise Server 10 SP2
- SUSE Linux Enterprise Server 11
```

查看 Linux 的版本。

```
[root@localhost oracle]# cat /proc/version
Linux version 2.6.9-78.ELsmp (brewbuilder@hs20-bc2-3.build.redhat.com) (gcc
version 3.4.6    20060404 (Red Hat 3.4.6-10)) #1 SMP Wed Jul 9 15:39:47 EDT 2008
```

（2）内核需求

```
On Asianux 2, Oracle Enterprise Linux 4, and Red Hat Enterprise Linux 4:
2.6.9 or later
```

通过如下指令查看当前的系统是否满足 Oracle 11g 对操作系统内核要求。

```
[root@localhost oracle]# uname -r
2.6.9-78.ELsmp
```

（3）包需求

操作系统：Asianux 2、Oracle Enterprise Linux 4 和 RedHat Enterprise Linux 4 对应的包如下所示。

```
binutils-2.15.92.0.2
compat-libstdc++-33-3.2.3
elfutils-libelf-0.97
elfutils-libelf-devel-0.97
gcc-3.4.6 *
gcc-c++-3.4.6*
glibc-2.3.4-2.41*
glibc-common-2.3.4
glibc-devel-2.3.4 *
glibc-headers-2.3.4 *
libaio-devel-0.3.105
libaio-0.3.105
libgcc-3.4.6
```

```
    libstdc++-3.4.6
    libstdc++-devel-3.4.6
    make-3.80
    numactl-0.6.4.i386
    pdksh-5.2.14
    sysstat-5.0.5
    unixODBC-2.2.11
    unixODBC-devel-2.2.11
```

这些包在安装光盘中，是必须安装的，下面在虚拟机环境下访问虚拟光驱中的包，并安装，如例子 2-2 所示。

例子 2-2　强制安装 RPM 包 glibc-headers-2.3.4-2.41.i386.rpm。

```
    [root@myoracle RPMS]# rpm -ivh glibc-headers-2.3.4-2.41.i386.rpm --nodeps
--force
    warning: glibc-headers-2.3.4-2.41.i386.rpm: V3 DSA signature: NOKEY, key ID
db42a60e
    Preparing...           ########################################### [100%]
      1:glibc-headers            ###########################################
[100%]
```

这里我们使用了--nodeps --force 选项， 这样就可以回避包之间的依赖性，可以直接安装成功。

例子 2-3　安装 RPM 包 binutils-2.15.92.0.2

```
    [root@localhost RPMS]# rpm -ivh binutils-2.15.92.0.2-25.i386.rpm
    warning: binutils-2.15.92.0.2-25.i386.rpm: V3 DSA signature: NOKEY, key ID
db42a60e
    Preparing...           ########################################### [100%]
          package binutils-2.15.92.0.2-25 is already installed
```

4. 创建所需的操作系统用户组和用户

使用 root 用户登录操作系统，创建安装 Oracle11g 数据库所需的用户组与用户。创建一个 Oracle 用户作为安装软件的所有者，该用户必须将 Oracle Inventory Group 作为其第一个用户组。这样 Oracle 用户就可以向中心目录写数据。Oracle 用户同时必须拥有 grid infrastructure home 的 OSDBA 组，这样数据库实例就可以登录到 ASM（Oracle 的自动存储管理），并且将 OSOPER 作为第二个用户组。

（1）下面我们创建这些用户组。

例子 2-4　创建各类用户组

```
    创建 OSDBA 用户组 DBA。
    # /usr/sbin/groupadd -g 502 dba
    创建 Oracle Inventory 用户组 oinstall.
    # /usr/sbin/groupadd oinstall
    创建 OSASM 用户组
    # /usr/sbin/groupadd -g 504 asmadmin
    创建 OSDBA 用户组
    # /usr/sbin/groupadd -g 506 asmdba
```

（2）创建 Oracle 软件拥有者（软件用户）。

例子 2-5 创建软件用户

```
# /usr/sbin/useradd -u 502 -g oinstall -G dba,asmdba,[oper] oracle
```

介绍参数。

```
-u :specifies the user ID.
-g :specifies the primary group
-G :specifies the secondary groups,which must include the OSDBA
group, and, if required, the OSOPER and ASMDBA groups
```

例子 2-6 创建用户密码

```
创建用户密码:
# passwd oracle
```

如果 Oracle 软件用户已经存在，或许需要修改使得所属的首选组和次选组分别为 oinstall 和 osdba(dba)，如下所示。

```
# /usr/sbin/usermod -g oinstall -G dba,asmdba[,oper] oracle
```

（3）修改 Oracle 软件安装用户的 Shell 限制。

```
打开的最大文件描述符          nofile   65536
单个用户可获得的最大进程数    nproc    16384
进程堆栈区的最大尺寸          stack    10240
在文件/etc/security/limits.conf 中增加如下参数设置。
oracle soft nproc 2047
oracle hard nproc 16384
oracle soft nofile 1024
oracle hard nofile 65536
```

在文件/etc/pam.d/login 中增加如下内容。

```
session required pam_limits.so
```

（4）配置内核参数。

编辑文件/etc/sysctl.conf，增加如下参数。

```
fs.aio-max-nr = 1048576
fs.file-max = 6815744
kernel.shmall = 2097152
kernel.shmmax = 536870912
kernel.shmmni = 4096
kernel.sem = 250 32000 100 128
net.ipv4.ip local port range = 9000 65500
net.core.rmem default = 262144
net.core.rmem max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048586
net.ipv4.tcp wmem = 262144 262144 262144
net.ipv4.tcp_rmem = 4194304 4194304 4194304
```

输入下面指令使得参数生效。

```
# /sbin/sysctl -p
```

可以通过如下指令确定参数正确。

```
/sbin/sysctl -a  --输出文件内容。
```

5. 所需的软件目录

（1）Oracle Base 目录

它指 Oracle 软件的顶层目录，Oracle 推荐的为/mount_point/app/software_owner，其中 mount_point 指挂接点目录，software_owner 指软件用户，如 Oracle，这样其 Base 目录为：

```
/u01/app/oracle
```

（2）Oracle Inventory 目录

Oracle Inventory 目录存储安装的软件目录，是一个 Oracle software installations 共享的目录，Oracle Universal Installer 创建这个目录，但是要求该目录具有安装软件的用户、用户组以及读写权限。在安装时，如果没有写权限会报错，读者可以根据错误再回头修改目录权限。

（3）Oracle Home 目录是安装数据库软件的目录

Oracle 推荐的 Oracle Home 目录格式：

```
oracle_base/product/11.2.0/dbhome_1
```

如 /u01/app/oracle/product/11.2.0/dbhome_1。下面我们依次创建这些目录，并修改环境变量，使得这些变量生效。

例子 2-7　创建 Oracle base 目录

```
# mkdir -p /u01/app/oracle
# chown -R oracle:oinstall /u01/app/oracle
# chmod -R 775 /u01/app/oracle
```

然后设置 ORACLE_BASE 环境变量 vi .bash_profile。

6. 创建数据库文件目录和快速恢复区目录

数据库文件目录:

```
# mkdir /mount point/oradata
# chown oracle:oinstall /mount_point/oradata
# chmod 775 /mount_point/oradata
The default location for Database file directory is $ORACLE BASE/oradata.
快速恢复区目录 (fast recovery area):
# mkdir /mount point/recovery area
# chown oracle:oinstall /mount_point/recovery_area
# chmod 775 /mount_point/recovery_area
```

上面我们就完成了操作系统需要的预配置任务，下面我们就可以进入数据库软件的安装了。

2.4.2　安装数据库软件

当然安装软件需要下载相应版本的 Oracle 数据库管理软件，这些软件 Oracle 都是免费下载使

用的，只要不是用于商业用途。读者只要登录 Oracle 官方网站，在下载（download）页面中选择需要的软件即可，这些软件都是压缩的文件，Oracle 11g 有两个压缩文件，解压后这些文件会解压到名为 database 的目录下。下面我们就安装解压后的文件。

01 安装 Oracle 11g 数据软件，如图 2-21 所示。

```
[oracle@myoracle data]$ cd database
[oracle@myoracle database]$ ls
doc  install  response  rpm  runInstaller  sshsetup  stage  welcome.html
[oracle@myoracle database]$ ./runInstaller
Starting Oracle Universal Installer...

Checking Temp space: must be greater than 80 MB.  Actual 4188 MB   Passed
Checking swap space: must be greater than 150 MB.  Actual 502 MB   Passed
Checking monitor: must be configured to display at least 256 colors.   Actual
16777216    Passed
Preparing to launch Oracle Universal Installer from /tmp/OraInstall2012-11-21
06-41-18AM. Please wait ...[oracle@myoracle database]$
```

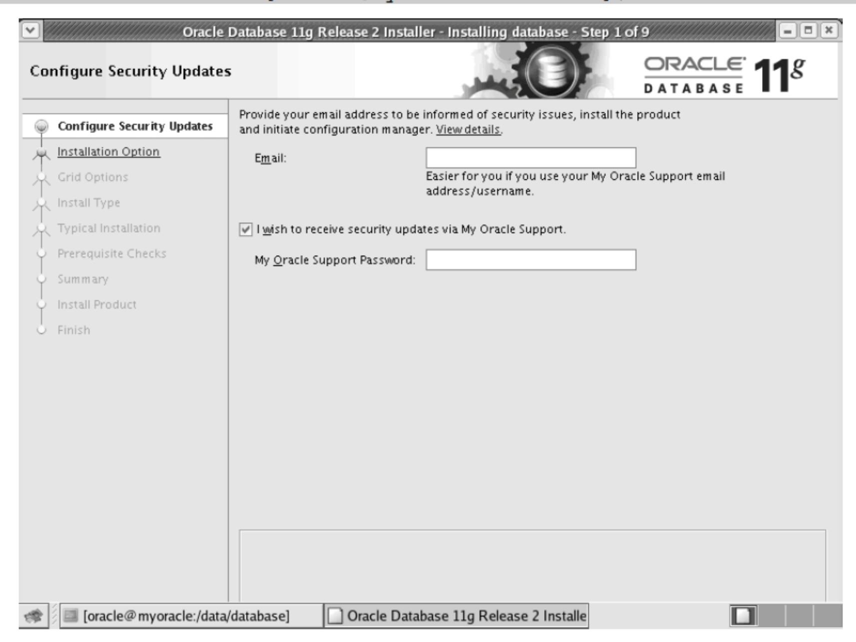


图 2-21　安装选项

02 单击下一步，或者直接单击回车键，就进入下一个页面，如图 2-22 所示。

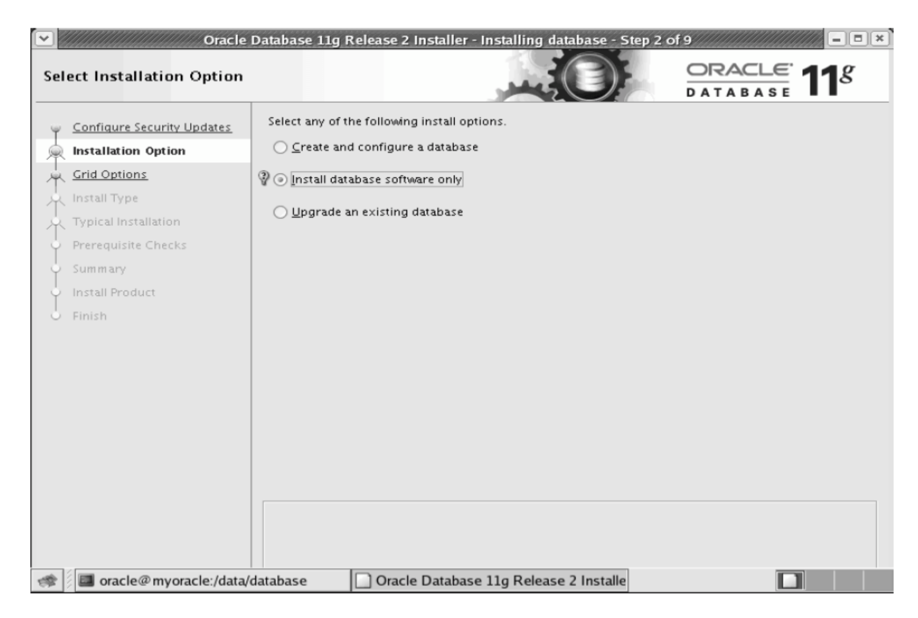


图 2-22　选择只安装数据库软件

此时，有 3 个选项，即创建并配置一个数据库、安装数据库软件以及升级数据库，我们选择第二个仅安装数据库软件，之后再创建数据库，这样的安排步骤对初学者是有益的。

03 单击下一步，选择单实例数据库，如图 2-23 所示。

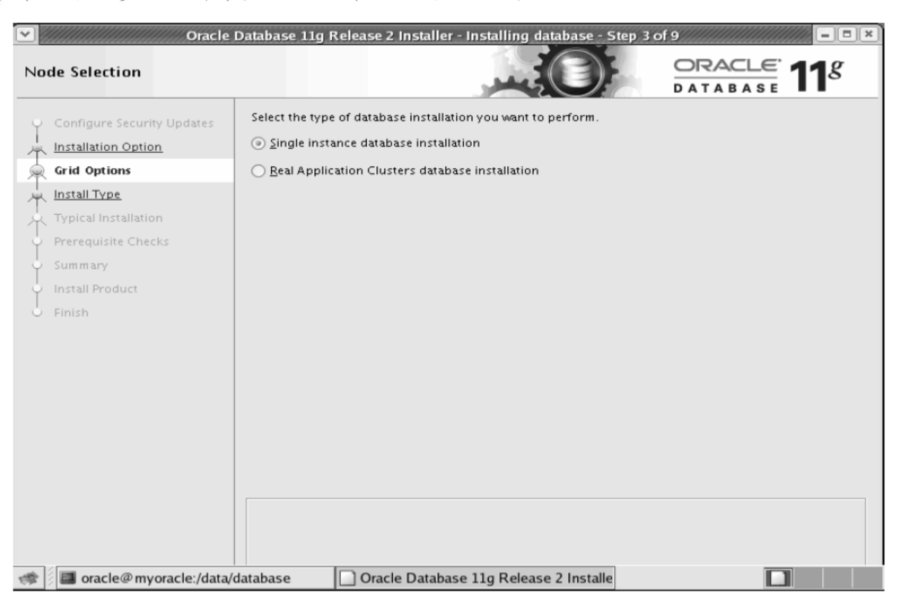


图 2-23　选择数据库安装类型

这里有两个选项，一个单实例数据库，一个是 RAC 数据库，显然，这里是单实例数据库，RAC 简单讲是多实例数据库环境，读者了解即可，这里我们选择第一个然后回车。

04 选择产品语言。

该语言是数据库软件运行的语言环境，我们选择 English，学习数据库大家还是习惯英语环境为好，如图 2-24 所示。

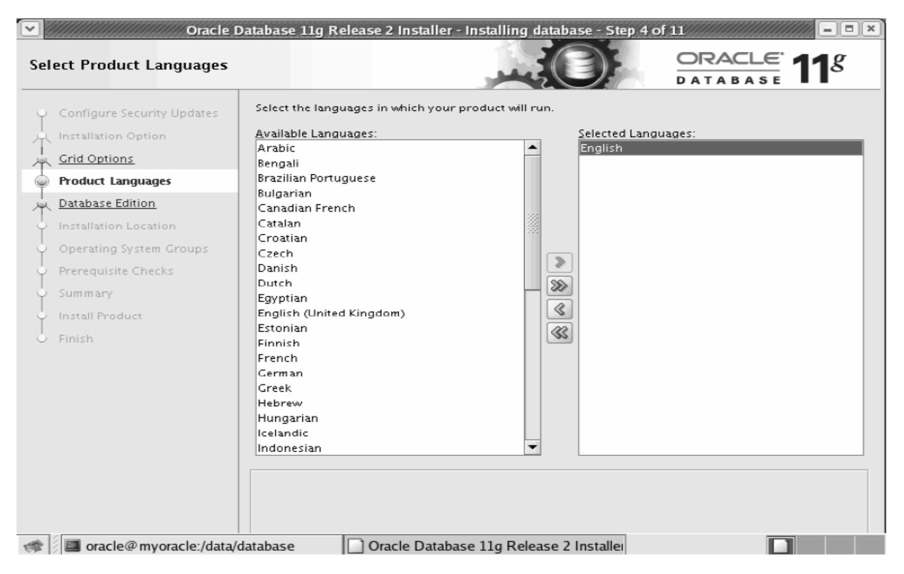


图 2-24 选择产品语言

05 单击下一步。选择数据库版本，此时提供了企业版、标准版。我们选择企业版，如图 2-25 所示。

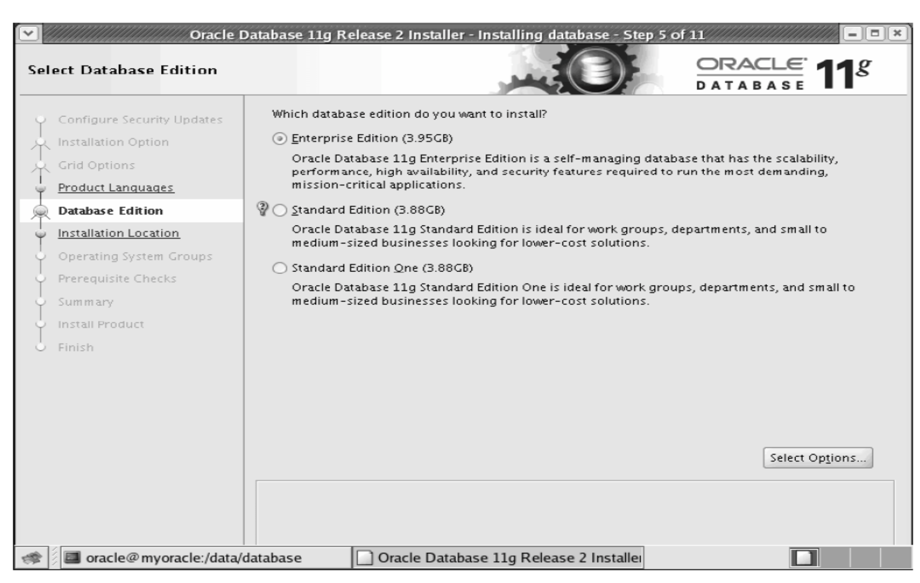


图 2-25　选择数据库版本

06 单击回车，选择数据库软件的安装目录，如图 2-26 所示。



图 2-26　选择安装目录

这里的 Oracle Base 是数据库软件的安装顶层目录，软件目录为 RDBMS 软件的安装目录。这些目录都是在环境变量中读取的。

07 创建 Inventory 目录，如图 2-27 所示。

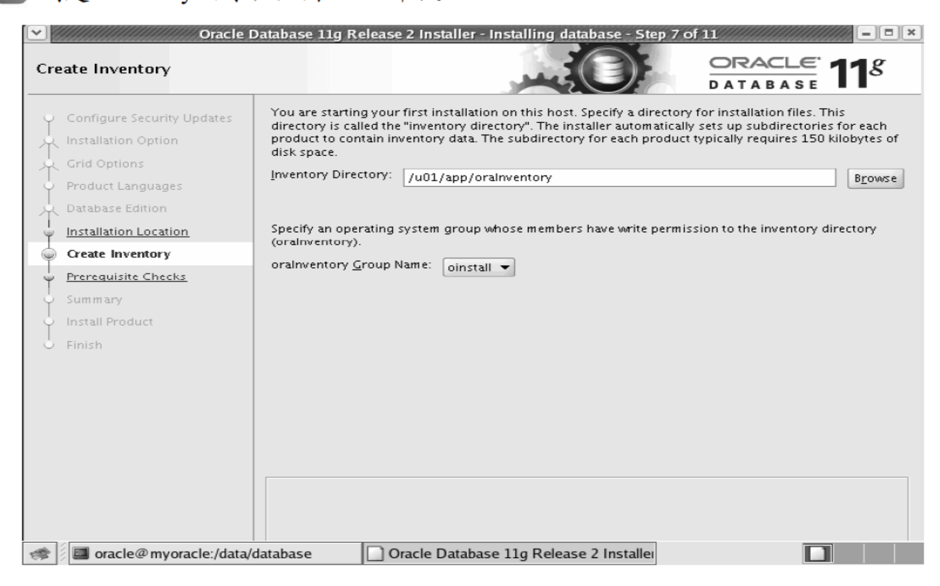


图 2-27　选择 Inventory 目录

这个目录有数据库软件自动安装，不需要提前设置。该目录所属的用户组为 oinstall。此时要求改用户必须具有对/u01/app 目录的读写权限，如果读者没有设置会报错。修改方式如下：

```
[root@myoracle ~]# chown -R oracle:oinstall /u01/app/
[root@myoracle ~]# chmod -R 755 /u01/app
```

08 选择 OSDBA 用户组，如图 2-28 所示。

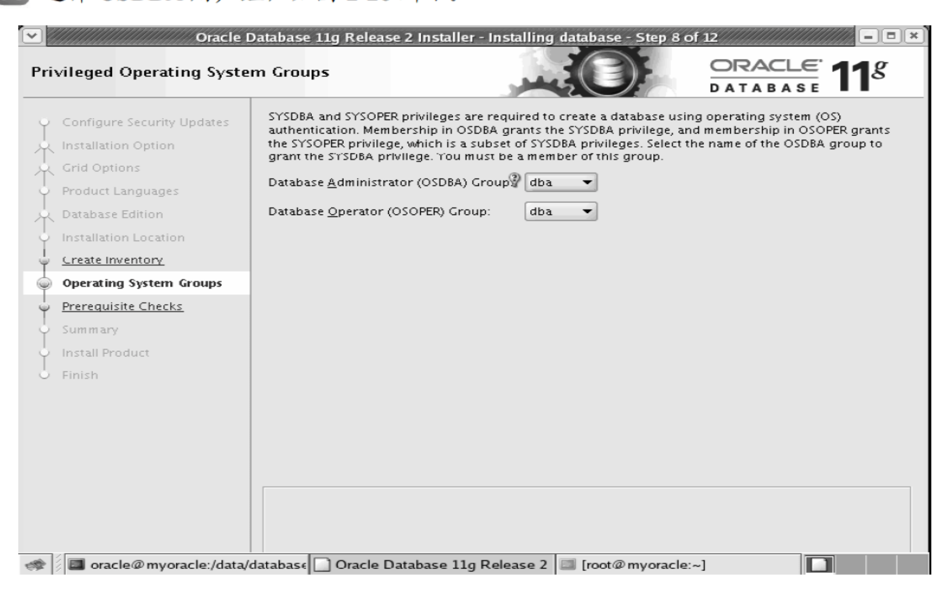


图 2-28　选择 OSDBA 用户组

09 预检查的过程，如果有需要的包没有安装的情况会提示，如图 2-29 所示。

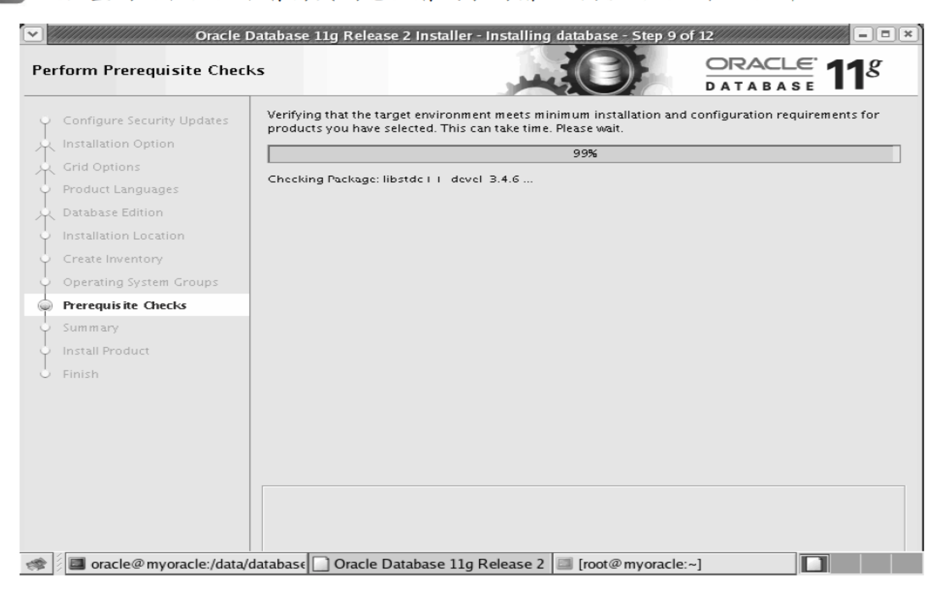


图 2-29　先决条件检查

不满足安装要求的系统包或者操作系统资源信息会做如图 2-30 所示的提示。

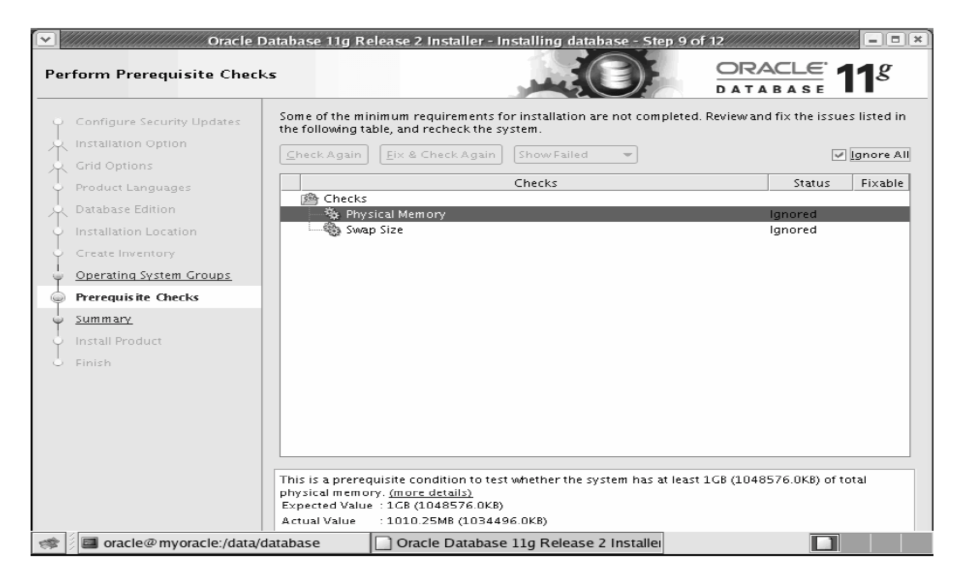


图 2-30　预安装检测结果

图 2-30 中提示系统的物理内存以及 SWAP 区的大小不满足要求，因为是虚拟环境我们暂时忽略这些，在实际系统中一定不能出现这样的问题，要配置足够的内存并设置足够大的 SWAP 区。接下来是全局设置信息和 Inventory 信息，如图 2-31 所示。

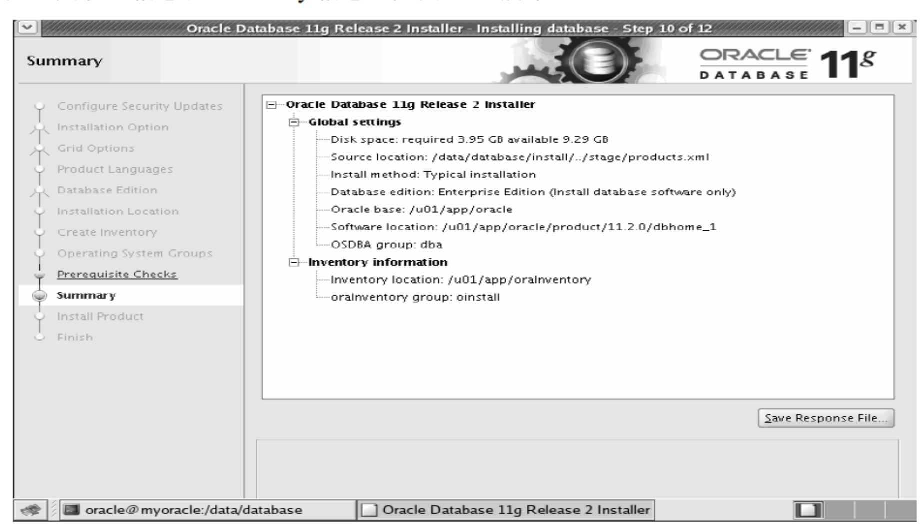


图 2-31　安装信息汇总

10 安装数据库软件。此时进入安装数据库软件的步骤，单击图 2-32 中的 Finish 按钮，开始安装数据库软件。

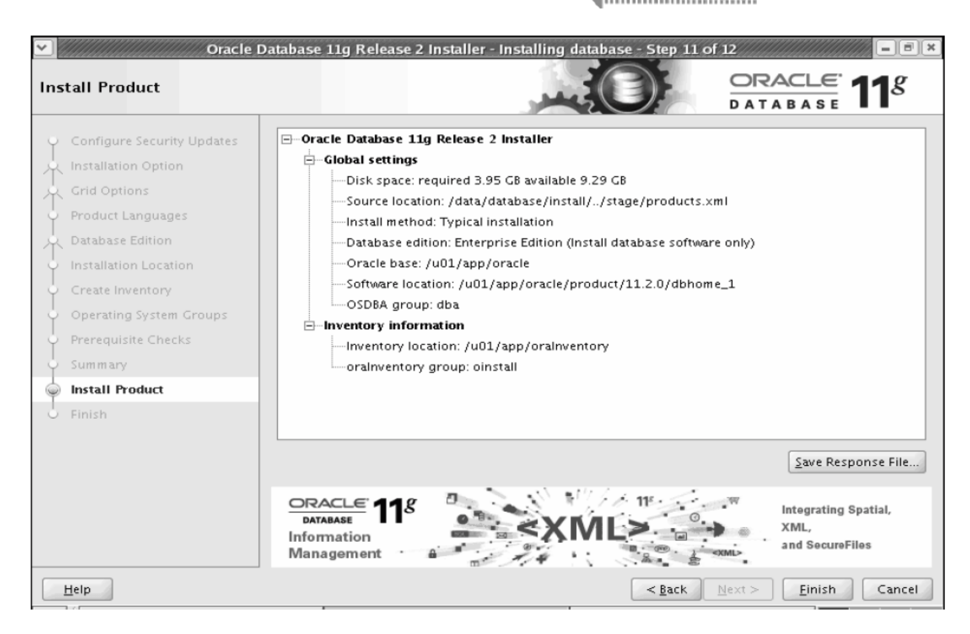


图 2-32　开始安装数据库软件

单击 Finish 按钮后该页面不发生变化，稍等片刻进入下个界面。

11 安装过程包括 4 个部分，它们组成了整个软件的安装过程，如图 2-33 所示。在这个步骤中如果 Link binaries 子步骤出现错误，一般是因为部分系统包没有安装成功。

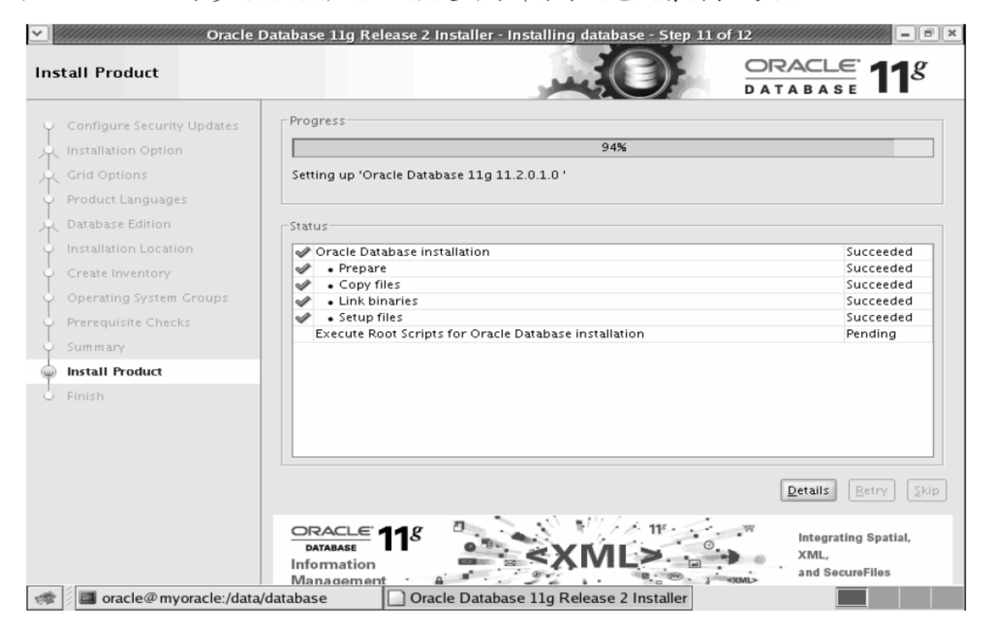


图 2-33　安装数据库软件过程

这个安装过程会经历几个步骤，任何一个步骤出现错误都会有提示，读者也可以通过日志文件监控安装的详细过程。

```
tail -f /u01/app/oraInventory/logs/installActions2012-11-21_06-48-23AM.log
```

12 执行脚本，如图 2-34 所示。

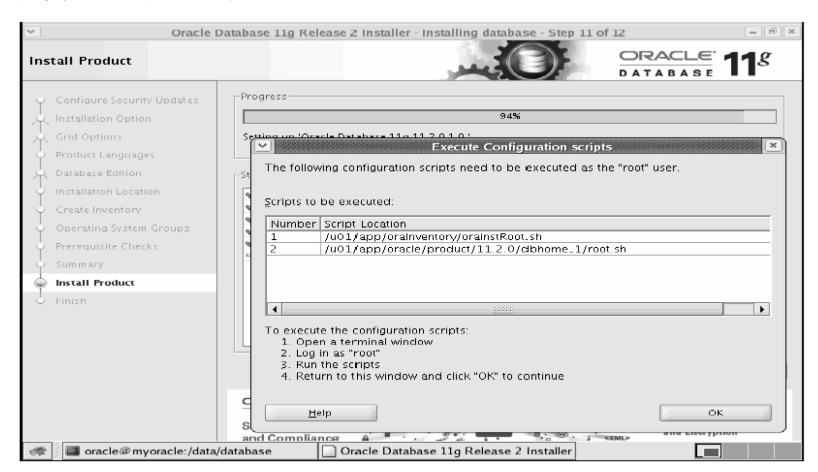


图 2-34　执行脚本

在安装 RDBMS 软件完毕后，会要求执行两个脚本，注意此时的脚本需要使用 ROOT 用户登录。

例子 2-8　执行脚本 orainstRoot.sh

```
[root@myoracle ~]# cd /u01/app/oraInventory/
[root@myoracle oraInventory]# ls
ContentsXML logs oraInst.loc oui
install.platform  oraInstaller.properties  orainstRoot.sh
[root@myoracle oraInventory]# ./orainstRoot.sh
Changing permissions of /u01/app/oraInventory.
Adding read,write permissions for group.
Removing read,write,execute permissions for world.

Changing groupname of /u01/app/oraInventory to oinstall.
The execution of the script is complete.
```

例子 2-9　执行脚本 root.sh

```
[root@myoracle dbhome 1]# pwd
/u01/app/oracle/product/11.2.0/dbhome_1
[root@myoracle dbhome_1]# ./root.sh
Running Oracle 11g root.sh script...

The following environment variables are set as:
    ORACLE_OWNER= oracle
    ORACLE HOME=  /u01/app/oracle/product/11.2.0/dbhome 1

Enter the full pathname of the local bin directory: [/usr/local/bin]:
   Copying dbhome to /usr/local/bin ...
   Copying oraenv to /usr/local/bin ...
   Copying coraenv to /usr/local/bin ...
```

```
Creating /etc/oratab file...
Entries will be added to the /etc/oratab file as needed by
Database Configuration Assistant when a database is created
Finished running generic part of root.sh script.
Now product-specific root actions will be performed.
Finished product-specific root actions.
```

13 当安装完成后，会显示如图 2-35 所示的对话框，提示 Oracle 软件安装成功。



图 2-35　安装成功

2.4.3　启动监听

监听是在数据库服务器端执行的一个进程，用于完成远程用户对数据库服务器的连接，在安装数据库过程中如果需要安装 EM（企业管理器），会要求监听已经启动，否则在安装过程中会提示错误。

01 使用 NETCA 启动监听配置，如图 2-36 所示。

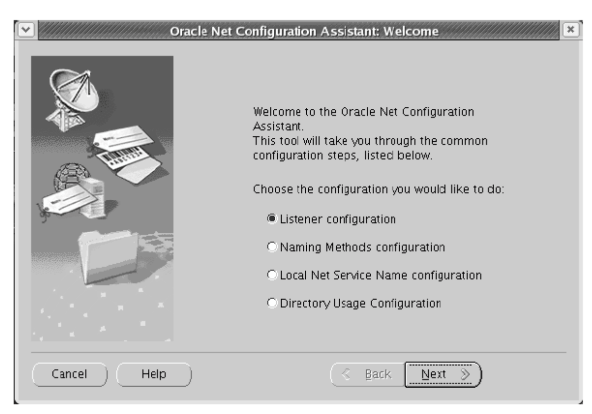


图 2-36　监听配置

02 增加监听器，监听的名称，这里采用默认命名，如图 2-37 所示。



图 2-37　增加监听

03 选择监听使用的协议，使用 TCP 传输控制协议，如图 2-38 所示。

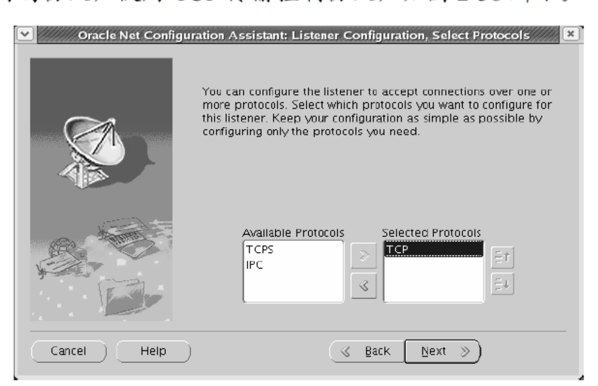


图 2-38　选择协议

04 选择监听使用的监听端口，监听毕竟是一个应用服务，我们这里使用默认值，如图 2-39、图 2-40 所示。

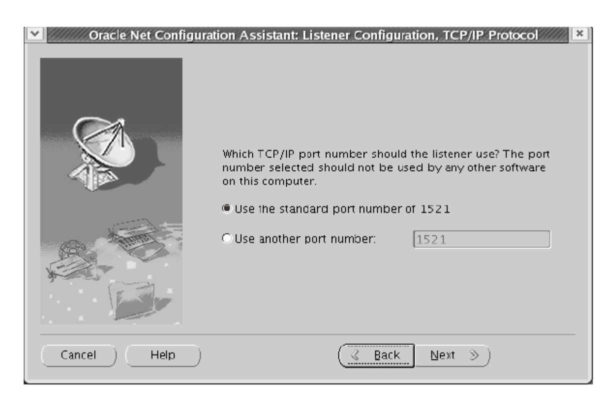


图 2-39　配置监听

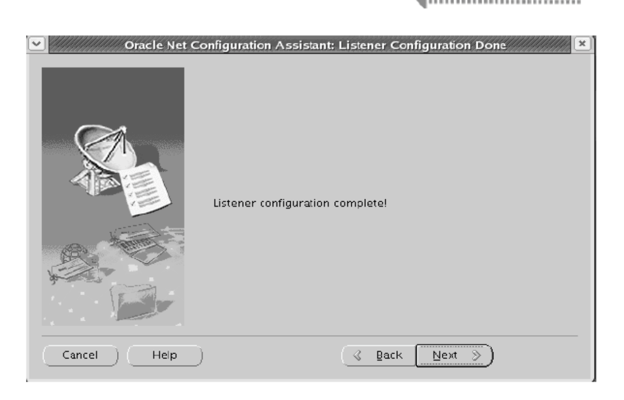


图 2-40　监听配置完成

05 单击 Finish 完成退出 NETCA，如图 2-41 所示。

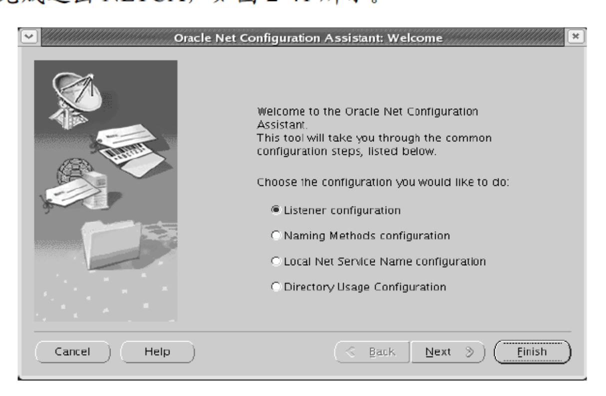


图 2-41　退出 NETCA

如下是 NETCA 在交互窗口执行的内容。

例子 2-10　NETCA 输出

```
[oracle@myoracle ~]$ netca

Oracle Net Services Configuration:
Configuring Listener:LISTENER
Listener configuration complete.
Oracle Net Listener Startup:
    Running Listener Control:
      /u01/app/oracle/product/11.2.0/dbhome 1/bin/lsnrctl start LISTENER
    Listener Control complete.
    Listener started successfully.
Oracle Net Services configuration successful. The exit code is 0
```

下面我们查看监听状态。

例子 2-11　显示监听状态

```
[oracle@myoracle ~]$ lsnrctl status

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 25-NOV-2012 11:37:37
```

```
Copyright (c) 1991, 2009, Oracle.  All rights reserved.

Connecting                                                                     to
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1521)))
STATUS of the LISTENER
------------------------
Alias                      LISTENER
Version                    TNSLSNR for Linux: Version 11.2.0.1.0 - Production
Start Date                 25-NOV-2012 11:35:36
Uptime                     0 days 0 hr. 2 min. 1 sec
Trace Level                off
Security                   ON: Local OS Authentication
SNMP                       OFF
Listener                              Parameter                              File
/u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
Listener                                 Log                                 File
/u01/app/oracle/diag/tnslsnr/myoracle/listener/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1521)))
The listener supports no services
The command completed successfully
```

以上输出很详细，读者需要自己阅读，如监听名称、监听文件，监听日志、监听连接设置。从最后的输出说明监听正常。

2.4.4 使用 DBCA 图形化工具建库

DBCA（DataBase Configuration Assistant）是 Oracle 数据库软件自带的配置助手，用于创建数据库、删除数据库、数据库配置任务。它是一个图像化工具，是初学者创建数据库的最简单的方法。因为我们在环境变量 PATH 中增加了$ORACLE_HOME/bin，所以在系统中输入 DBCA 指令就会启动 DBCA 配置助手，按照图形化的提示，一步步创建 Oracle 数据库，如图 2-42~图 2-57 所示。

```
[oracle@myoracle ~]$dbca
```

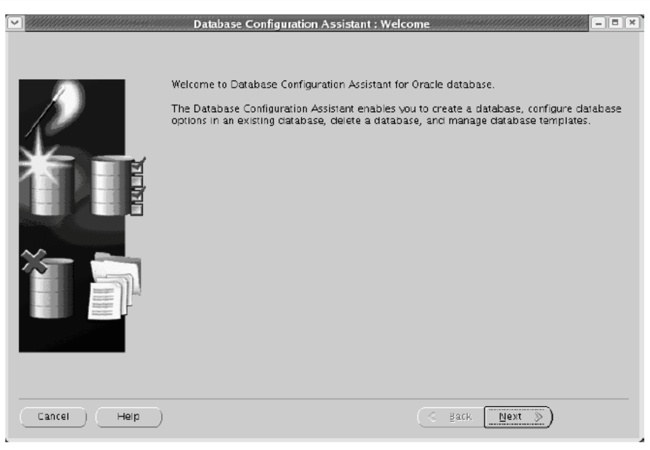


图 2-42　启动 DBCA

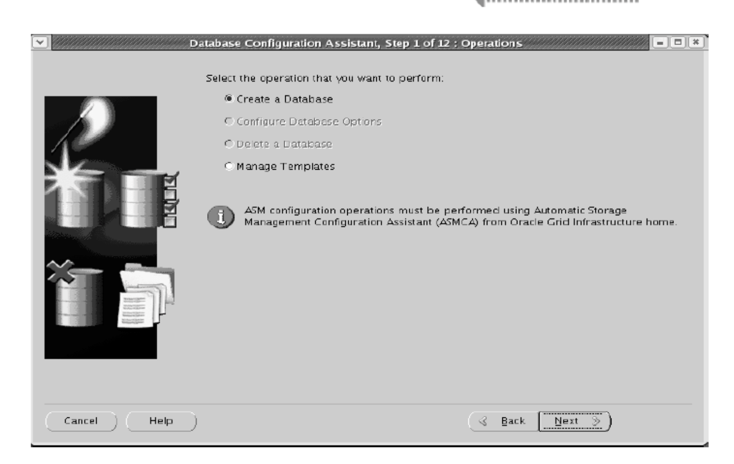


图 2-43　选择创建数据库

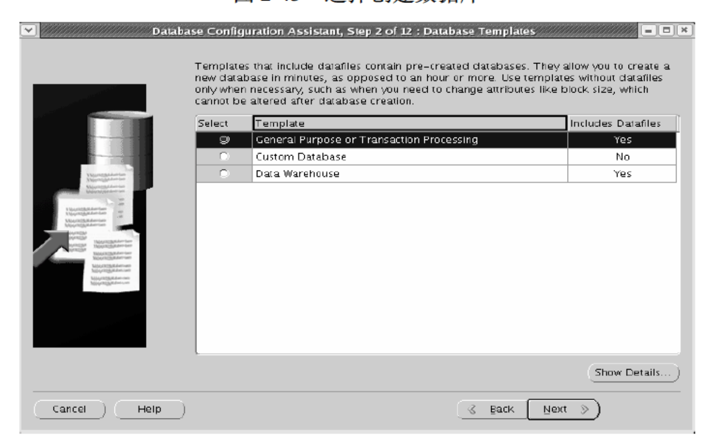


图 2-44　选择数据库类型



图 2-45　设置 Oracle SID

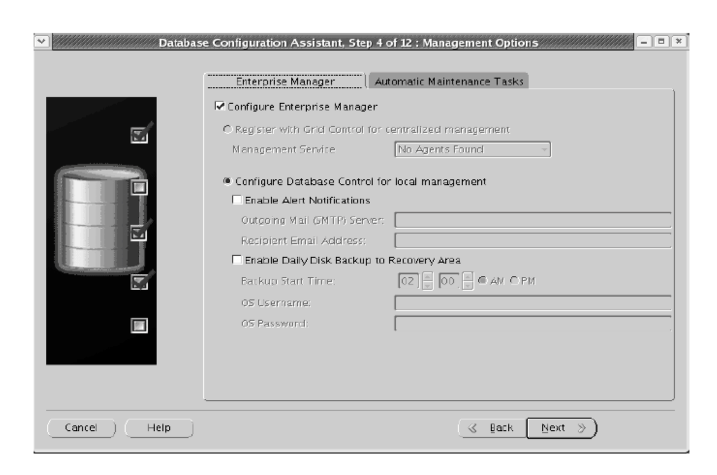


图 2-46　选择配置 EM

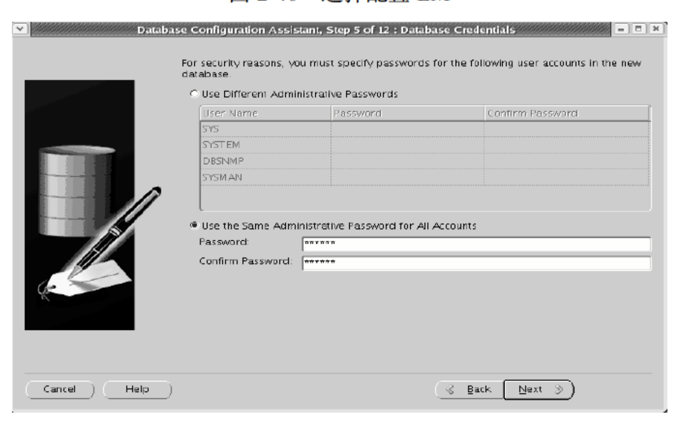


图 2-47　配置密码（Oracle）

在图 2-47 中，我们设置的密码是 oracle，提示没有满足 Oracle 要求的密码复杂度，我们按照错误提示的要求修改密码。下面重新输入密码（Oracle1234）如图 2-48 所示。

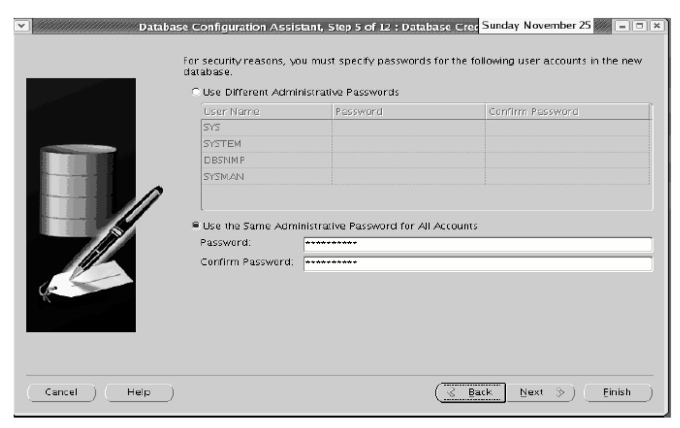


图 2-48　密码验证通过

单击下一步如下所示。

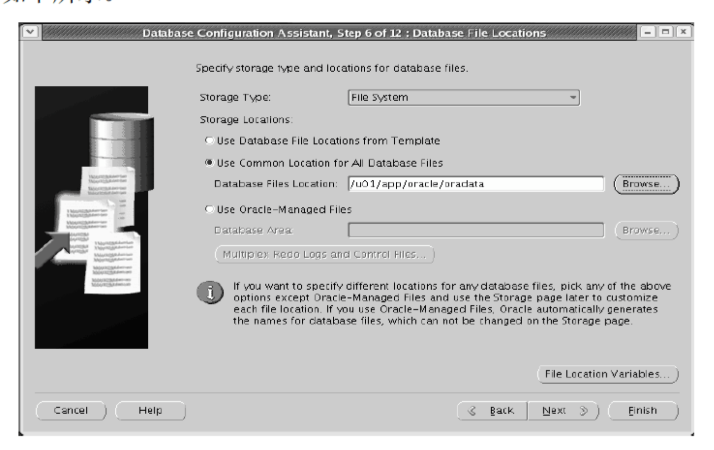


图 2-49　选择数据文件目录/u01/app/oracle/oradata

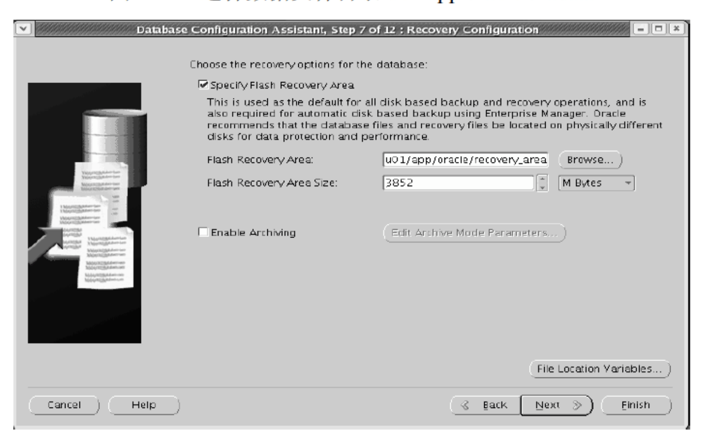


图 2-50　选择闪回恢复区目录

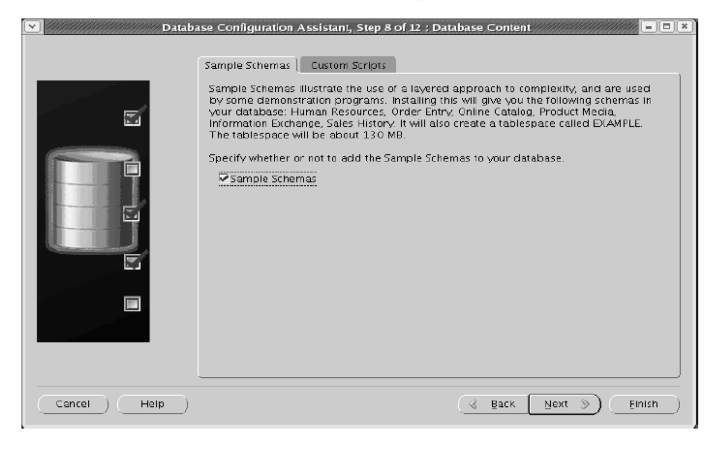


图 2-51　选择 SMPLE SCHEMAS

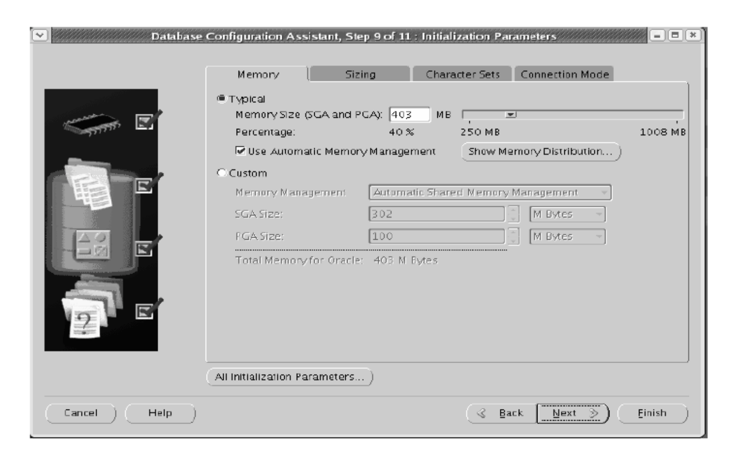


图 2-52　配置初始化参数（会根据系统资源自动配置）

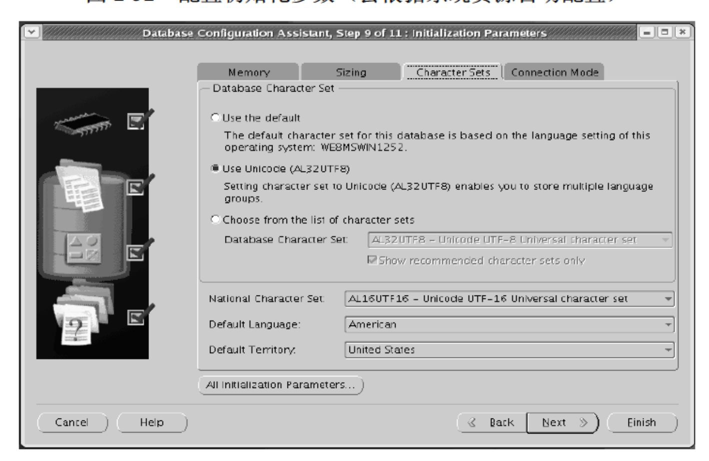


图 2-53　字符集选择 AL32UTF8

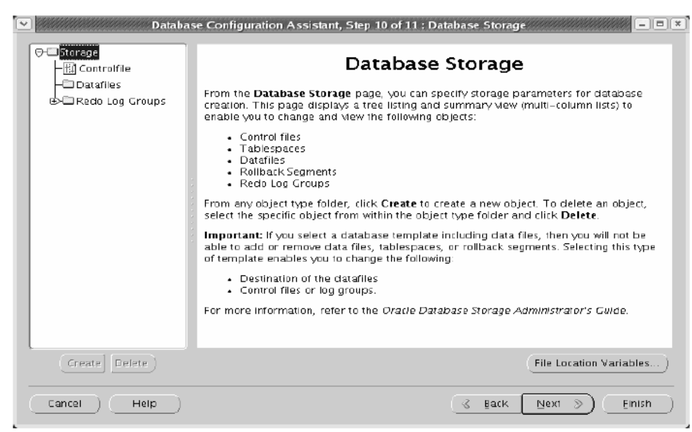


图 2-54　数据库文件的存储设置

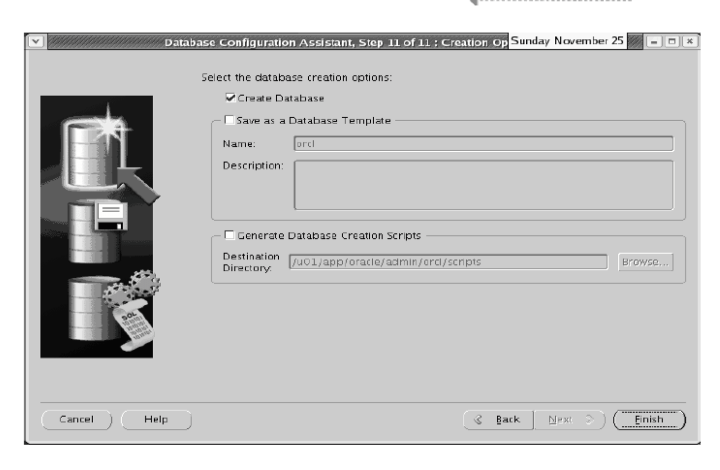


图 2-55　创建数据库

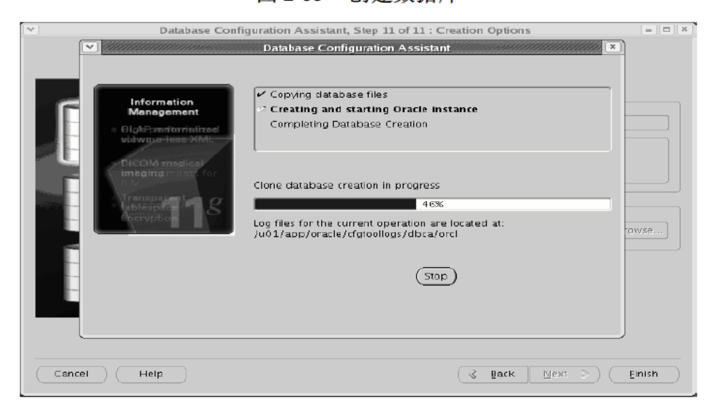


图 2-56　创建过程

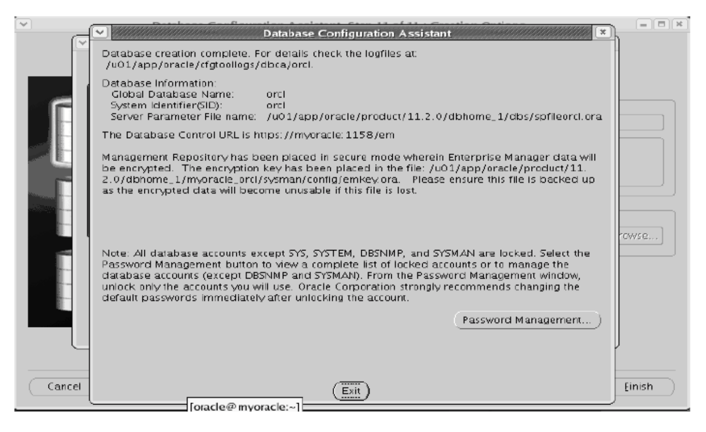


图 2-57　安装完毕

此时，数据库安装完成，这个对话界面提供了数据库的详细信息如数据库名、SID、服务器参数文件、启动 EM 的 URL 以及用户状态以及密码信息。

2.5 测试到数据库的连接

在数据库安装完毕后，需要测试到数据库的连接，我们在安装数据库软件的操作系统上使用SQL*Plus 工具来连接到数据库，此时为了直接连接到我们刚刚创建的数据库，重新编辑.bash_profile 文件，修改内容如例子 2-12 所示。

例子 2-12 修改.bash_profile 参数

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/11.2.0/dbhome_1
export ORACLE_SID=orcl
PATH=$PATH:$HOME/bin:$ORACLE_HOME/bin

export PATH
unset USERNAME
```

在原有内容的基础上增加 export ORACLE_SID=orcl 变量并使该文件生效，如下所示。

```
[oracle@myoracle ~]$ . .bash_profile
```

下面我们使用 SYS 用户登录来测试连接功能，此时必须使用 SYSDBA 角色。

例子 2-13 测试数据库连接

```
 [oracle@myoracle ~]$ sqlplus sys/oracle as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Sun Nov 25 12:17:37 2012

Copyright (c) 1982, 2009, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

然后查询动态性能视图 v$database，查看数据库的当前状态信息，如例子 2-14 所示。

例子 2-14 查询当前数据库的状态

```
SQL>     select     name     ,to_char(created,'yyyy-mm-dd     hh24:mi:ss') created,log_mode,open_mode
 from v$database;

NAME     CREATED              LOG_MODE        OPEN_MODE
------------------------ --------------------------- --------------------
ORCL     2012-11-25 11:59:32      NOARCHIVELOG   READ WRITE
```

显然，输出说明，数据库名为 ORCL，打开模式为读、写模式，数据库处于非归档模式。通过动态性能视图 v$database 的查询结果，说明当前使用 SQL*Plus 到数据库服务器的连接成功。

到目前为止，我们已经成功安装了数据库软件，创建了一个数据库，并启动了监听功能，我们就有了一个学习 Oracle 11g 数据库的环境。在以后的学习中就以这个环境为基础，所以希望读者认真完成安装过程。

2.6　删除数据库软件

Oracle 数据库的卸载在 11g 之前比较麻烦，存在卸载不干净的问题，这样会导致第二次安装的失败，随着 Oracle 升级到 11g，数据库卸载有了极大改善。因为它提供了一个卸载脚本，只要在执行该脚本前停止所有 Oracle 服务即可。该脚本名为 deinstall.bat。默认在目录$ORACLE_HOME/deinstall/下，运行该脚本即可自动完成数据库的卸载任务，最后只要手动删除相关文件夹即可。

下面是手工删除数据库软件的过程。之前如果建库成功，可以先使用 DBCA 删除数据库，再使用如下方式手工删除数据库软件。这个步骤比较繁琐，需要细心删除相关的文件和注册信息。

打开注册表（运行 regedit），删除如下要求删除的注册内容。

- 删除 HKEY_LOCAL_MACHINE/SOFTWARE/ORACLE 目录。
- 删除 HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Service 中所有以 Oracle 或者 OraWeb 开头的键。
- 删除 HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/Eventlog/application 中所有以 Oracle 开头的键。
- 删除 HKEY_CLASSES_ROOT 目录中所有以 Ora、Oracle、实例名或 EnumOra 为前缀的键。
- 删除 HKEY_CURRENT_USER/SOFTWARE/Microsoft/windows/CurrentVersion /Explorer/MenuOrder/Start Menu/Programs 中所有以 Oracle 开头的键。
- 删除 HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBCINST.INI 中除了 Microsoft ODBC for Oracle 键以外的、所有含 Oracle 的键。
- 删除环境变量中的 PATH CLASSPATH 中包含 Oracle 的键。
- 删除开始/程序中所有 Oracle 的图标。
- 删除所有与 Oracle 相关的目录。

2.7　本章小结

本章我们主要介绍了如何在 Windows 系统和 Linux 系统上安装 Oracle 11g 数据库软件，并创建数据库。介绍了我们在本书中会多次用到的 SCOTT 用户。对于 Windows 下安装数据库软件比较简单，只要读者的计算机硬件满足要求即可，而对于 Linux 系统则需要考虑诸如操作系统版本、内核参数、系统架构等要求，需要根据具体的操作系统而有所区别，本章最后给出了在 RedHat Enterprise Linux AS.V4.0.UPDATE.7 上安装 Oracle 11g 数据库软件的详细过程供读者参考。

第 3 章

数据库的启动与关闭

数据库的启动涉及一系列的文件读取和数据一致性检查等操作，但数据库启动时会首先启动数据库实例（Instance），在这个过程中数据库获得一些内存空间，并启动了必需的后台监控进程，而后会读取控制文件再进一步打开各种数据文件，最后完成数据库的启动任务。数据库的关闭执行和启动相反的过程。下面我们详细介绍数据库的启动和关闭过程，以及在启动和关闭过程中涉及的各类文件。

3.1 启动数据库

启动数据库需要读者以 DBA 用户登录，只有 DBA 用户才具有打开和关闭数据库的权限。启动数据库时，如果具有相应的权限，只需要输入 startup 指令就可以打开数据库。但是如果权限不够，则会出现如例子 3-1 所示的错误提示。

例子 3-1 权限不足时启动数据库

```
SQL> startup
ORA-01031: insufficient privileges
```

如果用户具有 DBA 权限，在输入 startup 指令后 Oracle 数据系统要执行一系列的复杂的操作，如读取参数文件、控制文件、打开数据文件，进行数据一致性检验等等，下面详细介绍数据库的启动过程。

3.1.1 数据库启动过程

数据库的启动过程涉及 3 个状态，这 3 个状态同时涉及 3 种文件，在每个状态数据库做不同的事情，同时这 3 个状态适用于数据库的不同维护要求。这 3 个状态如下：

- NOMOUNT 状态：该状态只打开了数据库实例，此时读取参数文件。
- MOUNT 状态：该状态 ORACLE 根据参数文件中控制文件的位置找到并打开控制文件，读取控制文件中的各种参数信息，如数据文件和日志文件的位置等，但是此时并不打开数据文件。
- OPEN 状态：该状态数据库将打开数据文件并进行一系列的检查工作，这些检查工作用于数据恢复。

图 3-1 中给出了一个数据库启动流程图，当数据库启动到 NOMOUNT 状态时，此时 Oracle 只打开数据库实例。在启动到 MOUNT 状态时，打开实例并读取控制文件。启动到 OPEN 状态则打开数据文件，日志文件等各类必需的数据库文件。在启动数据库时，我们可以直接启动数据库到 OPEN 状态，即打开数据库。但是这个过程仍然经历了我们介绍的 3 个状态过程，即：NOMOUNT→MOUNT→OPEN。在接下来的几节中我们分别介绍这 3 个状态。

图 3-1 说明数据库启动到不同状态的过程和涉及的操作。1 表示数据库启动到 NOMOUNT 状态，2 表示数据库启动到 MOUNT 状态，3 表示数据库启动到 OPEN 状态，方框表示启动到每一种状态涉及的操作，启动到 MOUNT 状态必然经历 NOMOUNT 状态，启动到 OPEN 状态必然经历 NOMOUNT 和 MOUNT 状态。

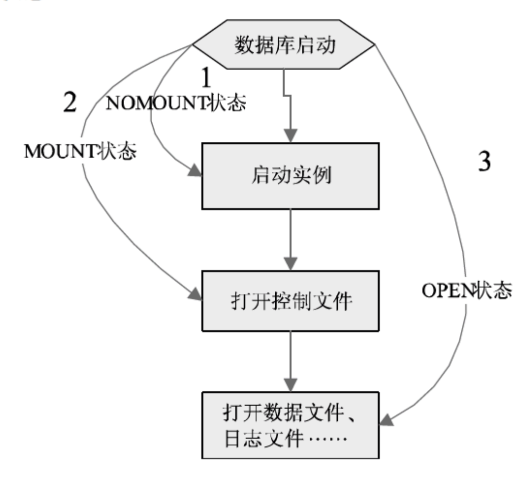


图 3-1　数据库启动流程图及相关状态

3.1.2　数据库启动到 NOMOUNT 状态

启动数据库到 NOMOUNT 状态时，会首先搜寻参数文件，读者只需要知道参数文件中存放了一些参数信息，如数据库缓冲区大小、重做日志缓冲区大小等。根据这些参数分配内存即 SGA，然后启动必需的后台进程，有 5 个后台进程是必须启动的，它们是 DBWR（数据库写进程）、LGWR（日志写进程）、SMON（系统监控进程）、PMON（进程监控进程）和 CKPT（检验点进程）。

该过程不涉及控制文件和数据文件，只需要一个参数文件就可以启动到 NOMOUNT 状态。启动到 NOMOUNT 状态的指令如例子 3-2 所示。

例子 3-2　启动到 NOMOUNT 状态

```
C:\Documents and Settings\oracle>sqlplus /nolog

SQL*Plus: Release 11.2.0.1.0 Production on 星期三 12 月 12 08:15:30 2012

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

SQL> connect / as sysdba
已连接到空闲例程。
SQL> startup
```

```
ORACLE 例程已经启动。

Total System Global Area  535662592 bytes
Fixed Size                  1375792 bytes
Variable Size             239075792 bytes
Database Buffers          289406976 bytes
Redo Buffers                5804032 bytes
数据库装载完毕。
数据库已经打开。
```

在例子 3-2 中启动数据库时，我们使用了 connect/as sysdba 方式连接数据库，这是一种操作系统认证方式，当然也可以使用在创建数据库使用的 SYS 用户登录。

数据库的启动过程记录在告警追踪文件中，该告警追踪文件中包括数据库启动的信息，它存放在参数 BACKGROUND_DUMP_DEST 定义的目录下，告警日志文件的名字为 alert_orcl.log，如果用户不知道，可以使用如下指令查询告警日志的存储目录，如例子 3-3 所示。

例子 3-3　查看存储告警追踪文件的参数值

```
SQL> show parameter background dump dest

NAME                    TYPE        VALUE
------------------------------------ ------------------------------
background_dump_dest    string      f:\app\oracle\diag\rdbms\orcl\orcl\trace
```

从上例的输出说明，记录告警追踪文件位于参数 background_dump_dest 指定的目录下，即在笔者的计算机上，告警追踪文件存储在 F:\app\oracle\diag\rdbms\orcl\orcl\trace 下，如图 3-2 所示。



图 3-2　Windows 平台上告警日志文件的存储目录

在图 3-2 的目录下，由于笔者的 SID 为 lin，所以告警文件名为 linALRT.log。下面查看一下该告警文件的内容，以确认数据库启动到 NOMOUNT 状态时的启动详细过程。下面的内容是从 linALRT.log 复制的，这部分记录说明了数据启动到 NOMOUNT 状态的启动过程。由于告警追踪文件记录太长，我们分开说明。

启动数据库实例的前期工作，如归档目录、启动审计等。

```
Wed Dec 12 08:15:44 2013
```

```
Starting ORACLE instance (normal)
LICENSE_MAX_SESSION = 0
LICENSE_SESSIONS_WARNING = 0
Picked latch-free SCN scheme 2
Using LOG_ARCHIVE_DEST_1 parameter default value as USE_DB_RECOVERY_FILE_DEST
Autotune of undo retention is turned on.
IMODE=BR
ILAT =27
LICENSE_MAX_USERS = 0
SYS auditing is disabled.
```

下面这部分记录内容是读取参数文件获得系统参数值，这些参数包括分配的 SGA 中非自动管理的内存组件大小，如大池和流池，控制文件的位置，数据库块大小以及还原表空间名。

```
Starting up:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options.
Using parameter settings in server-side spfile
F:\APP\ORACLE\PRODUCT\11.2.0\DBHOME_1\DATABASE\SPFILEORCL.ORA
System parameters with non-default values:
  processes                = 150
  memory_target             = 808M
  control_files             = "F:\APP\ORACLE\ORADATA\ORCL\CONTROL01.CTL"
  control_files             =
"F:\APP\ORACLE\FLASH_RECOVERY_AREA\ORCL\CONTROL02.CTL"
  db_block_size             = 8192
  compatible               = "11.2.0.0.0"
  db_recovery_file_dest     = "F:\app\oracle\flash_recovery_area"
  db_recovery_file_dest_size= 3852M
  undo_tablespace           = "UNDOTBS1"
  remote_login_passwordfile= "EXCLUSIVE"
  db_domain                = ""
  dispatchers              = "(PROTOCOL=TCP) (SERVICE=orclXDB)"
  local_listener            = "LISTENER_ORCL"
  audit_file_dest           = "F:\APP\ORACLE\ADMIN\ORCL\ADUMP"
  audit_trail              = "DB"
  db_name                 = "orcl"
  open_cursors              = 300
  diagnostic_dest           = "F:\APP\ORACLE"
```

下面这部分记录内容是启动后台的数据库管理进程。

```
Fri Oct 16 3:22:16 2009
PMON started with pid=2, OS id=4044
Fri Oct 16 3:22:16 2013
VKTM started with pid=3, OS id=328 at elevated priority
VKTM running at (20)ms precision
Fri Oct 16 3:22:16 2013
DIAG started with pid=4, OS id=2340
Fri Oct 16 3:22:16 2013
DBRM started with pid=5, OS id=3680
Fri Oct 16 3:22:16 2013
PSP0 started with pid=6, OS id=1504
```

```
Fri Oct 16 3:22:16 2013
DSKM started with pid=7, OS id=3040
Fri Oct 16 3:22:16 2013
DIA0 started with pid=8, OS id=1584
Fri Oct 16 3:22:16 2013
MMAN started with pid=7, OS id=3900
Fri Oct 16 3:22:16 2013
DBW0 started with pid=9, OS id=2832
Fri Oct 16 3:22:16 2013
LGWR started with pid=10, OS id=4036
Fri Oct 16 3:22:16 2013
CKPT started with pid=11, OS id=2652
Fri Oct 16 3:22:16 2013
SMON started with pid=3, OS id=3172
Fri Oct 16 3:22:16 2013
RECO started with pid=13, OS id=2804
Fri Oct 16 3:22:16 2013
MMON started with pid=14, OS id=1184
starting up 1 dispatcher(s) for network address
'(ADDRESS=(PARTIAL=YES)(PROTOCOL=TCP))'...
Fri Oct 16 3:22:16 2013
MMNL started with pid=15, OS id=2676
starting up 1 shared server(s) ...
ORACLE_BASE from environment = F:\app\Administrator
```

上述文件内容主要说明了数据库启动到 NOMOUNT 状态时的启动详细信息，首先 Oracle 读取参数文件分配内存区，读取其他如数据库块大小等参数，然后启动后台进程，但是没有打开控制文件的信息。

在数据库启动到 NOMOUNT 状态时，并不打开控制文件。在 Oracle 中查看控制文件存储目录的方法是使用视图 v$controlfile，这是一个动态视图，如果数据控制文件没有打开，则无法通过该动态视图查询到控制文件的存储目录。例子 3-4 测试在启动到 NOMOUNT 状态时，控制文件是否打开。

例子 3-4　测试在启动到 NOMOUNT 状态时控制文件是否打开

```
SQL> select *
  2  from v$controlfile;

未选定行
```

上例说明数据库没有打开控制文件。但是在 NOMOUNT 状态可以通过参数文件获得控制文件的位置，因为我们知道此时参数文件已经打开。在 NOMOUNT 状态，我们使用 v$parameter 动态视图获得控制文件的位置。如例子 3-5 所示。

例子 3-5　在 NOMOUNT 状态下使用 v$parameter 视图获得控制文件的位置

```
SQL> show parameter control_files;
NAME     TYPE  VALUE
------------------------------------------------ ----------------------
control files  string F:\APP\ORACLE\ORADATA\ORCL\CONTROL01.CTL,
               F:\APP\ORACLE\FLASH_RECOVERY_AREA\ORCL\CONTROL02.CTL
```

上例的输出说明当前数据库的控制文件的位置，我们在安装 Oracle11g 数据库时采用数据文件的默认位置，当然这个位置可以修改。

3.1.3　数据库启动到 MOUNT 状态

数据库在启动到 MOUNT 状态有两种方式，一是可以直接启动数据库到 MOUNT 状态，一是如果数据库已经启动到 NOMOUNT 状态，然后使用指令 alter database mount 把数据库切换到 MOUNT 状态，本例采用后者，在 3.1.2 节数据库启动到 NOMOUNT 状态后再启动到 MOUNT 状态，如例子 3-6 所示。

例子 3-6　在 NOMOUNT 状态下将数据库启动到 MOUNT 状态

```
SQL> ALTER database mount;

数据库已更改。
```

此时，数据库处于 MOUNT 状态，我们可以通过动态视图 v$controlfile 获得控制文件的存储目录。因为从 NOMOUNT 状态切换到 MOUNT 状态就是 Oracle 打开控制文件的过程。如例子 3-7 在 MOUNT 状态查看控制文件的存储目录。

例子 3-7　在 MOUNT 状态查看控制文件的存储目录

```
SQL> col name for a55
SQL> select status,name,block_size from v$controlfile;

STATUS  NAME                                                    BLOCK SIZE
------- ------------------------------------------------------- ----------
 F:\APP\ORACLE\ORADATA\ORCL\CONTROL01.CTL                          16384
 F:\APP\ORACLE\FLASH_RECOVERY_AREA\ORCL\CONTROL02.CTL  16384
```

因为我们打开了控制文件，所以可以通过动态数据字典视图 v$controlfile 来查看控制文件的状态或存储目录信息，以及控制文件本身的数据块大小和文件块大小等信息。在例子 3-7 中查询到的控制文件存储在 F:\APP\ORACLE\ORADATA\ORCL 目录下，并且每个控制文件的数据块大小都为 16384 B。

然后，我们在告警追踪文件中可以查看从 NOMOUNT 状态切换到 MOUNT 状态的过程。

```
alter database mount
Wed Dec 12 08:35:14 2012
Successful mount of redo thread 1, with mount id 1329600062
Database mounted in Exclusive Mode
Lost write protection disabled
Completed: alter database mount
```

记录显示在时间 Wed Dec 12 08:35:14 2012 执行了“alter database mount”，即将数据库状态转换到 MOUNT 状态的指令。

但是，此时数据库并没有打开，所以数据文件无法读取，我们用例子 3-8 说明在 MOUNT 状态下，数据库没有打开。

例子 3-8　在 MOUNT 状态下读取数据文件

```
SQL> select *
  2  from scott.dept;
from scott.dept
         *
ERROR 位于第 2 行:
ORA-0319: 数据库未打开: 仅允许在固定表/视图中查询
```

3.1.4　数据库启动到 OPEN 状态

启动数据库到 OPEN 状态，ORACLE 需要检验数据文件的头信息，进行点计数器检查和 SCN 检查来完成实例恢复，至于检查点计数器和 SCN 在控制文件与数据库启动一章中会做详细介绍。

数据库启动到 OPEN 状态，有两种方式，一是使用指令 startup open 或 startup（startup 的默认启动方式是启动到 OPEN 状态）直接启动到 OPEN 状态；二是如果数据库处于 NOMOUNT 或 MOUNT 状态，可以使用指令 alter database open 指令切换到 OPEN 状态。这里采用第二种方式。在数据库启动到 MOUNT 状态后切换到 OPEN 状态，如例子 3-9 所示。

例子 3-9　在数据库启动到 MOUNT 时打开数据库

```
SQL> alter database open;

数据库已更改。
```

数据库处于 OPEN 状态，此时我们可以查询数据库中的表数据，用户 SCOTT 的表都存储在 USERS 表空间中，该表空间是用户表空间存储数据文件，即用户的表都存储在这个表空间的数据文件中，如例子 3-10 所示。

例子 3-10　在打开数据库时查询数据库中的表

```
SQL> select *
  2  from scott.dept;

DEPTNO   DNAME           LOC
--------------- ---------------------------- -------------
       10 ACCOUNTING     NEW YORK
       20 RESEARCH       DALLAS
       30 SALES          CHICAGO
       40 OPERATIONS     BOSTON
```

显然，由于数据库切换到 OPEN 状态，所以数据文件打开，用户可以操作数据库中的数据文件中的各种对象，如查询指定用户表中的数据。

我们再来看看告警追踪文件 alert_orcl 中记录的该过程。

```
Wed Dec 12 08:40:12 2012
alter database open
Wed Dec 12 08:40:13 2012
Thread 1 opened at log sequence 36
  Current log# 3 seq# 36 mem# 0: F:\APP\ORACLE\ORADATA\ORCL\REDO03.LOG
Successful open of redo thread 1
MTTR advisory is disabled because FAST START MTTR TARGET is not set
Wed Dec 12 08:40:13 2012
SMON: enabling cache recovery
Successfully onlined Undo Tablespace 2.
```

```
Verifying file header compatibility for 11g tablespace encryption..
Verifying 11g file header compatibility for tablespace encryption completed
SMON: enabling tx recovery
Database Characterset is ZHS16GBK
No Resource Manager plan active
Starting background process QMNC
Wed Dec 12 08:40:17 2012
QMNC started with pid=20, OS id=2908
Completed: alter database open
Wed Dec 12 08:40:25 2012
Starting background process CJQ0
Wed Dec 12 08:40:25 2012
CJQ0 started with pid=25, OS id=3612
Wed Dec 12 08:40:27 2012
db recovery file dest size of 3852 MB is 10.62% used. This is a
user-specified limit on the amount of space that will be used by this
database for recovery-related files, and does not reflect the amount of
space available in the underlying filesystem or ASM diskgroup.
```

3.2　关闭数据库

关闭数据库同启动数据库顺序正好相反，其基本思路是首先关闭各种数据文件，关闭打开的控制文件，然后关闭实例，数据库的启动经历了 NOMOUNT、MOUNT 和 OPEN 3 个阶段，数据库的关闭正好相反，它经历了 CLOSE、DISMOUNT 和 SHUTDOWN 3 个阶段。

3.2.1　数据库关闭过程

正如启动数据库，我们可以分步骤地启动数据库，从 NOMOUNT→MOUNT→OPEN，而关闭数据库也可以分步骤关闭，从 CLOSE→DISMOUNT→SHUTDOWN，如图 3-3 所示。下面我们分步骤介绍关闭数据库的过程，此时假设数据库已经正常启动并可以访问。

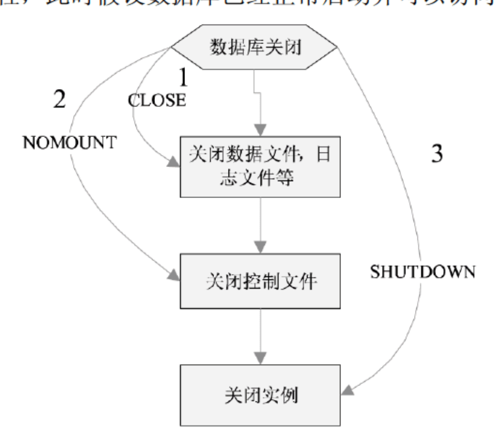


图 3-3　数据库关闭过程和涉及的操作

1. CLOSE 数据库

我们用例子 3-11 说明如何 CLOSE 数据库。

例子 3-11　关闭数据库

```
SQL> ALTER DATABASE close;

数据库已更改。
```

此时，可以通过查看告警文件的方式，观察这个过程如下所示。

```
Wed Dec 12 08:44:00 2012
alter database close
Warning: ALTER DATABASE CLOSE is not a publicly supported command.
Wed Dec 12 08:44:00 2012
SMON: disabling tx recovery
Wed Dec 12 08:44:00 2012
Stopping background process CJQ0
Stopping background process QMNC
All dispatchers and shared servers shutdown
CLOSE: killing server sessions.
CLOSE: all sessions shutdown successfully.
SMON: disabling cache recovery
Wed Dec 12 08:44:03 2012
Shutting down archive processes
Archiving is disabled
Archive process shutdown avoided: 0 active
Thread 1 closed at log sequence 36
Successful close of redo thread 1
Completed: alter database close
Wed Dec 12 08:45:05 2012
idle dispatcher 'D000' terminated, pid = (17, 1)
```

2. DISMOUNT 数据库

我们用例子 3-12 说明如何 DISMOUNT 数据库。

例子 3-12　将数据库切换到 DISMOUNT 状态

```
SQL> ALTER DATABASE dismount;

数据库已更改。
```

在告警文件中会有如下记录。

```
Wed Dec 12 08:46:54 2012
alter database dismount
Completed: alter database dismount
```

3. SHUTDOWN 数据库

SHUTDOWN 数据库会关闭数据库实例，使用该 SHUTDOWN 关闭数据库，Oracle 会做一些处理工作，如断开连接、回滚数据等，这些操作依赖于 SHUTDOWN 所选择的参数，在接下来我们会介绍使用这些参数的例子。现在我们继续第二个步骤，用例子 3-13 说明 SHUTDOWN 数据库。

例子 3-13　关闭数据库

```
SQL> shutdown
```

```
ORA-01507: 未安装数据库


ORACLE 例程已经关闭。
```

此时，查看告警文件观察，Oracle 记录的 SHUTDOWN 操作记录。

```
Wed Dec 12 08:47:52 2012
Shutting down instance (normal)
Shutting down instance: further logons disabled
Stopping background process MMNL
Stopping background process MMON
License high water mark = 5
All dispatchers and shared servers shutdown
ALTER DATABASE CLOSE NORMAL
ORA-1507 signalled during: ALTER DATABASE CLOSE NORMAL...
ARCH: Archival disabled due to shutdown: 1090
Shutting down archive processes
Archiving is disabled
Archive process shutdown avoided: 0 active
Wed Dec 12 08:47:56 2012
Stopping background process VKTM:
ARCH: Archival disabled due to shutdown: 1090
Shutting down archive processes
Archiving is disabled
Archive process shutdown avoided: 0 active
Wed Dec 12 08:47:58 2012
Instance shutdown complete
```

最后这个关闭行为其实就是关闭数据库实例的过程，因为在前两个关闭步骤中已经停止了对数据文件的操作并关闭了控制文件。

3.2.2　数据库关闭的几个参数及其含义

其实，在关闭数据库时，经常使用的是 SHUTDOWN 指令，选择不同的参数满足不同的关闭数据库的要求。

在 3.2.1 节启动数据库的第三个步骤中，我们说过 SHUTDOWN 数据库需要进行一些额外的操作，如断开连接、回滚数据等，而这些操作依赖于 SHUTDOWN 参数的选择。它有 4 个参数，即 NORMAL、IMMEDIATE、TRANSACTIONAL 和 ABORT。下面分别进行介绍。

- SHUTDOWN NORMAL

这种方式是 SHUTDOWN 数据库的默认方式，如果用户输入 SHUTDOWN，则默认采用 NORMAL 参数，这种方式关闭数据库时，不允许新的数据库连接，只有当前所有的连接都退出时才会关闭数据库，这是一种安全关闭数据库的方式，但是如果有大量的用户连接，则需要较长时间关闭数据库。

- SHUTDOWN IMMEIDATE

这种方式可以较快且安全地关闭数据库，是 DBA 经常采用的一种关闭数据库的方式，此时 ORACLE 会做一些操作，中断当前事务，回滚未提交的事务，强制断开所有用户连接，执行检查

点把脏数据写到数据文件中。虽然参数 IMMEDIATE 有立即关闭数据库的含义，但是它只是相对的概念，如果当前事务很多且业务量很大，则中断事务、回滚数据，以及断开连接的用户都需要时间。

当关闭数据库时，笔者推荐使用 SHUTDOWN IMMEDIATE 指令。

- SHUTDOWN TRANSACTIONAL

使用 TRANSACTIONAL 参数时，数据库当前的连接继续执行，但不允许新的连接，一旦当前的所有事务执行完毕，则关闭数据库。

显然通常情况下，在生产数据库系统中这种方式也不会快速关闭数据库，因为如果当前的某些事务一直执行，或许会用几天时间关闭数据库。

- SHUTDOWN ABORT

这是一种很不安全地关闭数据库的方式，最好不要使用该方式关闭数据库。SHUTDOWN ABORT 关闭数据库时，ORACLE 会断开当前的所有用户连接，拒绝新的连接，断开当前的所有执行事务，立即关闭数据库。使用这种方式关闭数据库，当数据库重启时需要进行数据库恢复，因为它不会对未完成事务回滚，也不会执行检查点操作。

3.3 本章小结

本节主要讲了数据库启动和关闭的过程，以及在启动和关闭过程中涉及的各种文件操作。在启动数据库时，涉及 3 种状态，即 NOMOUNT、MOUNT 和 OPEN，通常使用 STARTUP 指令，选择不同的参数就可以启动数据库到不同的状态，来满足不同的业务需要。关闭数据库和启动数据库正好相反，关闭数据库也涉及 3 个过程，即 CLOSE、DISMOUNT 和 SHUTDOWN。用户可以选择使用不同的参数，这些参数是 NORMAL、IMMEDIATE、TRANSACTIONAL 和 ABORT。在关闭数据库时，最好使用 SHUTDOWN IMMEDIATE 方式，这种方式安全且相对较快，不是万不得已不要使用 SHUTDOWN ABORT 方式，因为这种方式会造成数据丢失，恢复数据库也需要较长时间。读者需要理解数据库启动和关闭涉及的几个状态和相关的文件操作，这样在数据库启动中出现问题时就可以较好地处理故障点。

第 4 章

Oracle数据库体系结构

本章将讲解 Oracle 体系结构，它是全书非常重要的一章，这章的内容对于 Oracle 各个版本的数据库变化很小，所以是读者需要认真学习和体会的一章。

学习 Oracle 数据库，一开始从宏观上把握它的物理组成、文件组成和各种管理进程，对于进一步的学习起到很好的指导作用，不会使读者觉得在学习某一部分知识时，只见树木不见森林。相反，如果读者基本掌握或者理解了 Oracle 数据库体系结构的知识，就可以从更高的角度看待以后学习的内容。希望读者在学习本章时要细细体会和揣摩。本章将仅仅围绕数据库体系结构图详细介绍每一个数据库系统结构组件。

4.1　Oracle体系结构概述

体系结构是对一个系统的框架描述，是设计一个系统的宏观工作。打个比喻，就如建一座大楼需要设计图纸一样，根据建筑框架图的要求，“严格”施工就可以建造一座功能完善，质量可靠的建筑。即使大楼建好以后，我们仍然依据设计图纸来找到几乎每一个功能部件。

数据库系统结构设计了整个数据库系统的组成和各部分组件的功能，这些组件各负其职、相互协调完成数据库的管理和数据维护工作。

4.2　Oracle数据库体系结构

为了满足生产数据库需求，Oracle 设计了如图 4-1 所示的体系结构，该体系结构包括实例（Instance）、数据库文件、用户进程（User process）、服务器进程（Server process）以及其他文件，如参数文件（Parameter file）、密码文件（Password file）和归档日志文件（Archived log file）等。

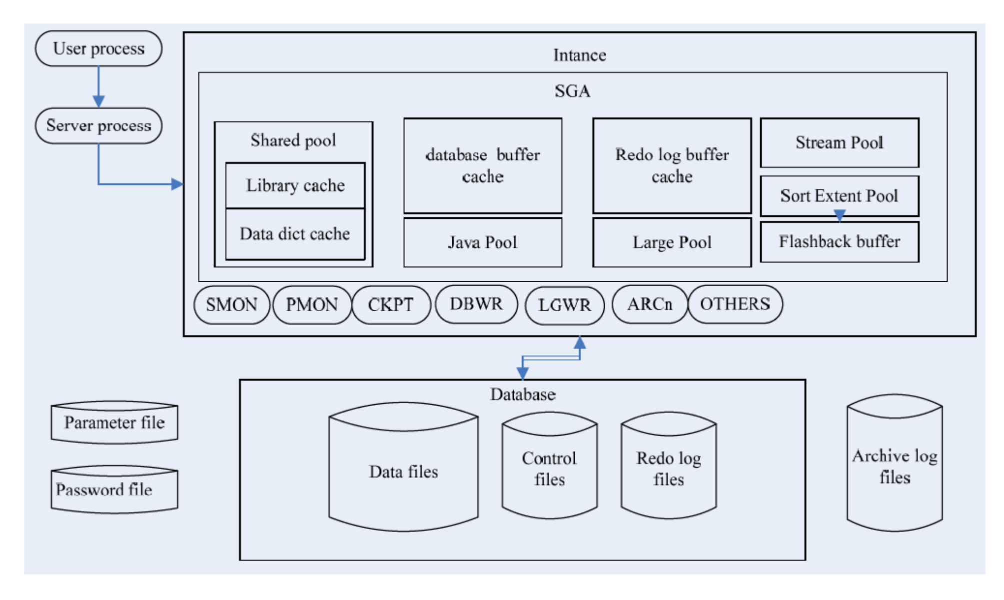


图 4-1　Oracle 数据库体系结构

其中，数据库实例包括 SGA（系统全局区）和一系列后台管理、监视进程，数据库包括三种文件：数据文件（Data files）、控制文件（Control files）和重做日志文件（Redo log files）。数据库实例和数据库是 Oracle 数据库体系结构的核心部分，DBA 很重要的工作就是维护实例和数据库本身的正常工作。

本节将依次介绍数据库服务器和实例、数据库的物理结构、密码文件和参数文件以及归档日志文件。

4.2.1　Oracle 服务器和实例

Oracle 服务器和实例是非常重要的两个概念，这里的服务器不仅仅是物理概念，还包括系统进程，而实例则是 DBA 经常维护的对象。下面依次介绍实例和服务器。

- Oracle 实例（instance）

Oracle 实例就是由一些内存区和后台进程组成。实例的组成如图 4-2 所示，从实例的组成图可以看到实例由 SGA 和一些后台进程组成，这些内存区包括数据库高速缓存、重做日志缓存、共享池、流池以及其它可选内存区（如 Java 池），这些池也称为数据库的内存结构，我们在 4.4 节会详细介绍。后台进程包括系统监控进程（SMON）、进程监控（PMON）、数据库写进程（DBWR）、日志写进程（LGWR）、检验点进程（CKPT）、其它进程（SMON，如归档进程、RECO 进程等），这些数据库系统进程忠于职守、相互协作完成数据管理任务。

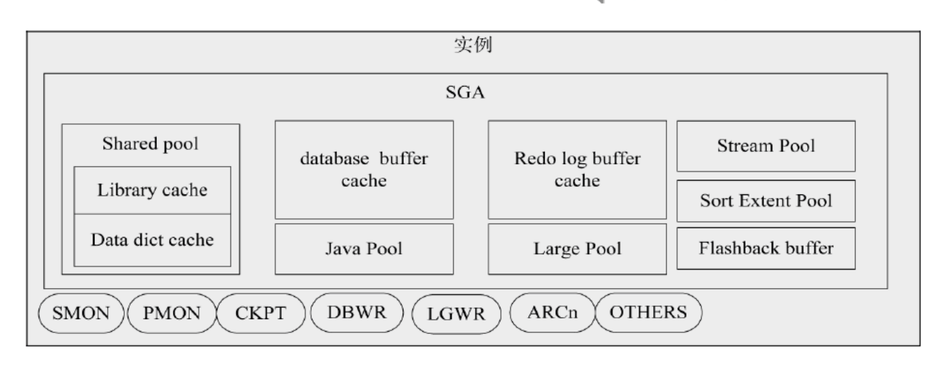


图 4-2　Oracle 实例（Instance）组成图

要访问数据库必须先启动实例，实例启动时先分配内存区，然后再启动后台进程，后台进程执行库数据的输入、输出以及监控其它 Oracle 进程。在数据库启动过程中有五个进程是必须启动的，它们是系统监控进程（SMON）、进程监控（PMON）、数据库写进程（DBWR）、日志写进程（LGWR）、检验点进程（CKPT），否则实例无法创建。数据库启动过程我们可以在告警日志（alertSID.ora）中看到详细的过程。



在实践中，为了方便会通过数据库工具实现数据库在计算机重启时自动启动，如果用户安装了其它占用大量内存的应用软件，可能会造成数据库启动失败，此时往往是因为内存不足，操作系统无法为 Oracle 分配 SGA。必须的五个进程也无法启动。

- Oracle 服务器（server）

Oracle 服务器由数据库实例和数据库文件组成，它就是我们经常说的数据库管理系统（DBMS）。数据库服务器的组成如图 4-3 所示。

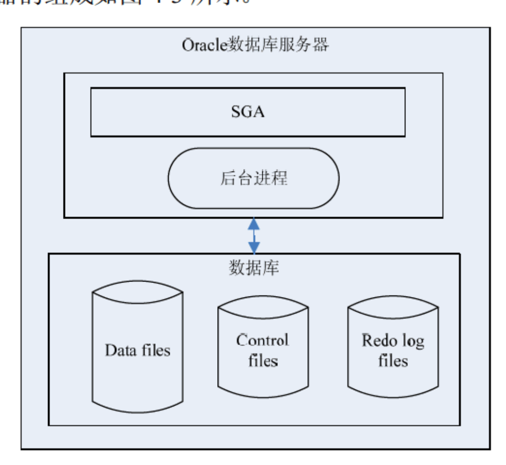


图 4-3　Oracle 服务器组成

数据库服务器除了维护实例和数据库文件外，还在用户建立与服务器的连接时启动服务器进程并分配 PGA。

4.2.2 Oracle 数据库物理结构（文件组成）

我们都知道，数据库是运行在操作系统之上的，数据库的最终目的就是存储和获取相关的数据，这些数据实际上存储在操作系统文件中，这些操作系统文件组成 Oracle 数据库物理结构。

Oracle 数据库的物理结构就是指数据库中的一系列操作系统文件，Oracle 数据库由 3 类文件组成。

- 数据文件（data files）：数据文件包含数据库中的实际数据，是数据库操作中数据的最终存储位置。
- 控制文件（control files）：包含维护数据库和验证数据库完整性的信息，它是二进制文件。
- 重做日志文件（redo files）：重做日志文件包含数据库发生变化的记录，在发生故障时用于数据恢复。

4.2.3 参数文件、密码文件和归档日志文件

虽然参数文件、密码文件和归档日志文件不是 Oracle 的数据库文件，但却是 Oracle 数据库不可少的 3 个文件。

- 参数文件（parameter file）：参数文件中定义了数据库实例的特性。在参数文件中包含为 SGA 中内存结构分配空间的参数，如分配数据库高速缓冲区的大小等，参数文件是正文文件，可以使用操作系统文本编辑器查看，如在 Windows 操作系统中使用记事本工具。
- 密码文件（password file）：密码文件授予用户启动和关闭数据库实例，在刚安装数据库时，Oracle 的默认用户名和密码就存储在密码文件中，Oracle 可以借此判断用户的操作权限。
- 归档日志文件（archive log files）：归档日志文件是日志文件的脱机备份，在发生故障后进行数据恢复时可能使用该文件。

4.3 数据库连接（connection）与会话（session）

连接与会话是 Oralce 数据库中容易混淆的两个概念，它们是紧密相关的两个概念。下面讲解它们的区别，给出实际的例子以帮助读者更好地理解它们的概念。

4.3.1 数据库连接（connection）

连接指用户进程与数据库服务器之间的通信途径，一个连接可以有多个对话。Oracle 提供了 3 种数据库连接方式，以满足用户不同的连接需求，3 种连接方式如下。

- 基于主机的方式（Host-Based）:此方式中，服务器和客户端运行在同一台计算机上，用户可以直接连接数据库服务器。
- 基于客户机-服务器的方式（Client-Server）:该方式中数据库服务器和客户端运行在不同的计算机上，客户通过网络连接数据库服务器。在 DBA 的日常维护中，会经常使用这种方

式访问数据库，实现数据库的远程维护。

- 客户-应用服务器-数据库服务器方式（Client-Application Server-Server）:这种方式称为三层访问模式，用户首先访问应用服务器，然后由应用服务器连接数据库服务器，应用服务器就如一个中介，完成客户和数据库的交互。在很多应用系统中，客户的应用程序往往通过三层方式访问数据库，如应用服务器为 IIS 或 Apache 服务器等。

4.3.2　会话（session）

会话指一个明确的数据库连接。在 4.3.1 节中讲了用户与数据库服务器建立连接的 3 种方式，一旦用户采用其中一种连接方式，就把这样的连接称为一个会话。

如用户通过某种工具（如 SQL*Plus）在专用连接的情况下访问数据库，在输入的用户名和密码经过服务器验证后，服务器就会自动创建一个与该用户进程相对应的服务器进程，二者是一对一的关系，这里服务器进程就像用户进程的代理，代替用户进程向数据库服务器发出各种请求，并把从数据库服务器获得的数据返回给用户进程。

但用户退出或发生异常时（操作系统重启）会话结束。



刚才指出“专有连接”的概念，专有连接是一种连接类型，指用户和服务器进程之间是一对一的关系。而在共享服务器配置的情况下，多个用户进程可以同时共享服务器进程，此时就不是专有连接，而是多对一的关系。

一个用户可以并发地建立多个会话，例子 4-1 就是用户 SYS 同时在专有连接的情况下建立两个会话的例子。

例子 4-1　用户 SYS 通过专有连接建立两个会话

```
SQL>SELECT serial#,username,status,server,process,program,logon time
  2* FROM v$session
SERIAL# USERNAME STATUS  SERVER   PROCESS   PROGRAM   LOGON TIME
----------- --------------------- --------------- ----------------------
     1          ACTIVE   DEDICATED 2172      ORACLE.EXE     17-5 月 -13
     1          ACTIVE   DEDICATED 528       ORACLE.EXE     17-5 月 -13
     1          ACTIVE   DEDICATED 2188      ORACLE.EXE     17-5 月 -13
     1          ACTIVE   DEDICATED 408       ORACLE.EXE     17-5 月 -13
     1          ACTIVE   DEDICATED 1424      ORACLE.EXE     17-5 月 -13
     1          ACTIVE   DEDICATED 1244      ORACLE.EXE     17-5 月 -13
     1          ACTIVE   DEDICATED 2264      ORACLE.EXE     17-5 月 -13
     3 SYS       ACTIVE   DEDICATED 540:1644   sqlplus.exe      17-5 月-13
     7 SYS       ACTIVE   DEDICATED 1296:1848  sqlplusw.exe     17-5 月-13

已选择 9 行。
```

在例子 4-1 的输出中我们同时使用两个 SQLPLUS 工具连接数据库，并且使用同一个用户 SYS，我们看到最后两行显示有两个活跃（ACTIVE）的会话，一个是使用 sqlplus.exe 程序建立的，一个是使用 sqlplusw.exe 程序建立的。



在上述查询中，只是演示一个用户可以建立多个连接，使用相同或不同的工具登录。至于v$session（数据字典视图）读者暂且把它看成是一张表，表中存储了当前会话的信息，如属性USERNAME是用户登录名，属性PROGRAM是用户登录工具（一个用户进程）。

图4-4清晰地说明了连接与会话之间的区别和联系。

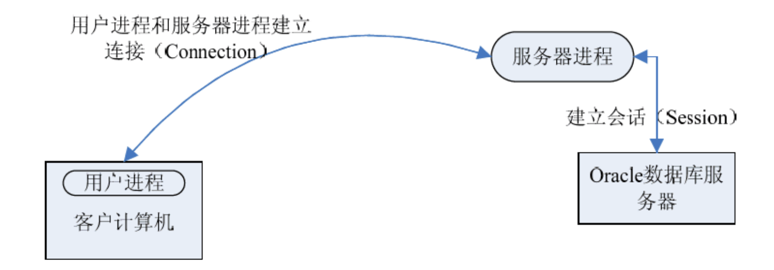


图4-4　连接与会话示意图



一个连接可以对应多个对话，连接仅仅是一种通信途径，如通过Socket建立通信，但是一个用户可以启动多个进程通过一个连接建立多个对话，这里服务器进程就像是用户进程的代理一样，与服务器交互完成数据的各种操作。

4.4　Oracle数据库内存结构

Oracle的内存结构由两大部分组成，一个是SGA，一个是PGA。PGA称为程序全局区，程序全局区不是实例的一部分，当服务器进程启动时，才分配PGA。而SGA称为系统全局区，它是数据库实例的一部分，当数据库实例启动时，会首先分配系统全局区，在系统全局区中包含几个重要的内存区，即数据库高速缓存（Database buffer cache）、重做日志缓存（Redo log buffer cache）、共享池（Shared pool）、大池（Large pool）和Java池（Java pool）。接下来的几个小节将详细介绍Oracle数据库的内存结构。

4.4.1　共享池（Shared pool）

Oracle引入共享池的目的就是共享SQL或PL/SQL代码，即把解析得到的SQL代码的结果在这里缓存，其中PL/SQL代码不仅在这里缓存，同时在这里共享。共享池由两部分组成，即库高速缓存（Libray cache）和数据字典高速缓存（Data dict cache），如图4-5所示。

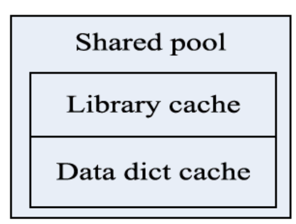


图 4-5　共享池的组成

- *库高速缓存*

库高速缓存存储了最近使用过的 SQL 和 PL/SQL 语句。当然它的容量是有限的，Oracle 采用一种 LRU（least recently used）算法管理库高速缓存，算法的基本思想是把一段时间内没有被使用过的语句清除，一旦缓冲区填满，算法把最近最少使用的执行计划和解析树从库高速缓存中清除。显然库高速缓存设置得越大，就可以共享更多的 SQL 或 PL/SQL 代码，但是 Oracle 并没有设计直接设置库高速缓存的指令，只能通过设置共享池的大小间接地更改，而共享池是 SGA 的一部分，所以共享池不能超过 SGA 的大小。下面两个例子分别设置和查看共享池的大小。

例子 4-2　设置共享池的大小

```
SQL> alter system set shared_pool_size = 16M;

系统已更改。
```

通过例子 4-3 验证修改结果。

例子 4-3　查看共享池的大小

```
SQL> show parameter shared_pool_size;

NAME                                TYPE         VALUE
-------------------------------- ---------------------------------------
shared_pool_size                     big integer      16777216
```

- *数据字典高速缓存*

顾名思义，该缓存区是与数据字典相关的一段缓冲区。在数据字典高速缓冲区中存储了数据文件、表、索引、列、用户、权限信息和其它一些数据库对象的定义。在 SQL 语句的解析阶段，数据库服务器需要这些信息来解析用户名和用户的访问权限。如果 Oracle 缓存了这些信息，无疑缩短了查询的相应时间。

数据字典缓存也称为字典缓存或者行缓存，无论称呼如何读者只需要理解它的作用就是把相关的数据字典信息放入缓存以缩短查询的响应时间。

同样数据字典高速缓存的大小取决于共享池尺寸的大小。如果设置得太小，在查询需要数据字典信息时，Oracle 将不断地访问数据字典表来获得所需的信息，由于数据字典也是存储在磁盘上的一类数据文件，频繁地磁盘 I/O 无疑降低了数据库的查询速度。如果需要设置字典高速缓存的大小，需要通过设置参数 shared_pool_size 间接实现。

4.4.2 数据库高速缓冲区（Database buffer cache）

数据库高速缓冲中存储了最近从数据文件读入的数据块信息或用户更改后需要写回数据库的数据信息，此时这些没有提交给数据库的更改后的数据称为脏数据。当用户执行查询语句如 select * from dept 时，如果用户查询的数据块在数据库高速缓存中，Oracle 就不必从磁盘读取，而是直接从数据库高速缓存中读取，显然物理读取的速度比从内存读取的速度慢很多，这些缓存的数据由 LRU 算法管理。可见 Oracle 设计的各种缓存目的基本相同，就是提高查询速度，减少用户查询的响应时间。Oracle 使用 LRU 算法管理库缓冲区，把最近没有使用的数据库从库高速缓存中删除，为其他的查询数据块保留空间。

Oracle 使用参数 DB_BLOCK_SIZE 和 DB_BLOCK_BUFFERS 设置库高速缓存的大小，DB_BLOCK_SIZE 是 Oracle 数据块的大小，而 DB_BLOCK_BUFFERS 是数据库的个数，二者的乘积就是库高速缓存的大小。例子 4-4 查询 Oracle 数据块的大小。

例子 4-4　查询数据库块的大小

```
SQL> show parameter db_block_size;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------------
db_block_size                            integer      8192
```



用这种方式设置数据库高速缓存的大小需要重启数据库才能生效，db_block_size 的值是 8192 B（即 8KB）。

在 Oracle 9i 及以上版本中提供了一个 DB_CACHE_SIZE 参数来设置 Oracle 数据库高速缓存区的大小，该参数可以动态更改，之后可以通过查询指令查看更改后的参数。

接下来用例子 4-5 查询 Oracle 11g 中数据库高速缓存的大小。

例子 4-5　查询数据库高速缓存的大小

```
SQL> show parameter db_cache_size;

NAME                                 TYPE        VALUE
-------------------------------------------------------- ------------------
db_cache_size                          big integer     0
```

因为在 Oracle 11g 中，SGA 为数据库服务器自动管理，所以该参数值为 0，当然在运行 Oracle 11g 数据库时，数据库高速缓存一定已分配好，我们可以使用 show sga 指令查看数据库高速缓冲区的分配的内存大小，如例子 4-6 所示。

例子 4-6　查询数据库高速缓存的大小

```
SQL> show sga;

Total System Global Area  535662592 bytes
Fixed Size                  1334380 bytes
Variable Size             260047764 bytes
```

```
Database Buffers          268435456 bytes
Redo Buffers                5844992 bytes
```



上述指令查询 SGA 的分配情况，其中 Database Buffers 为数据库缓存区的大小。我们更改的值 32MB，而显示的值为 33554432B，二者一致，说明我们修改成功（32MB=32×1024×1024B=33554432B）

虽然在 Oracle 11g 中数据库高速缓存的大小自动管理，但是用户可以设置该数据库组件的大小，如例子 4-7 所示。

例子 4-7　动态设置数据库高速缓冲区大小

```
SQL> alter system set db_cache_size = 200M;

系统已更改。
```

在 Oracle 中引入了 Buffer Cache Advisory Parameter 参数，其目的是让 Oracle 对于数据库缓冲区的内存分配提供一些建议。

下面介绍缓冲区顾问参数（Buffer Cache Advisory Parameter）的作用，缓冲区顾问用于启动或关闭统计信息，这些信息用于预测不同缓冲区大小导致的不同行为特性。对于 DBA 可以参考这些统计信息，基于当前的数据库工作负载设置优化的数据库高速缓存。

缓存顾问通过初始化参数 DB_CACHE_ADVICE 启动或关闭顾问功能，该参数有 3 个状态。

- OFF：关闭缓存顾问，不分配缓存顾问的工作内存。
- ON：打开缓存顾问，分配工作内存。
- READY：打开缓存顾问，但不分配缓存顾问的工作内存。

例子 4-8　当前缓存顾问的状态

```
NAME                                 TYPE        VALUE
------------------------------------ ----------------- -----------------
db_cache_advice                           string       ON
```

在上述输出中可以看出，参数 db_cache_advice 的值为 ON，所以默认是打开缓存顾问的。例子 4-9 演示了如何设置缓存顾问为关闭状态，其实就是通过设置参数 db_cache_advice 实现的。

例子 4-9　关闭数据库高速缓存顾问

```
SQL> alter system set db cache advice = off;

系统已更改。
```

在更改了缓存顾问状态后，通过例子 4-10 查看当前的缓存顾问状态，以验证更改结果。

例子 4-10　查看数据库高速缓存顾问状态

```
SQL> show parameter db_cache_advice;

NAME                                 TYPE        VALUE
----------------------------------------- ------------------------------
```

```
db_cache_advice                    string      OFF
```

当然我们的目的是使用缓存顾问，所以需要再次将参数 db_cache_advice 的值设置为 ON。

```
SQL> alter system set db cache advice = on;

系统已更改。
```

在设置顾问缓存为开启状态后，Oracle 开始统计与设置数据库缓存相关的建议信息，可以通过动态性能视图 v$DB_CACHE_ADVICE 查看缓冲区的建议信息，如例子 4-11 所示。

例子 4-11　查看与设置数据库高速缓冲区相关的信息

```
SQL> col id for 99
SQL> SELECT id, name, block size, size for estimate, buffers for estimate
  2  from v$db cache advice;

 ID NAME        BLOCK_SIZE SIZE_FOR_ESTIMATE BUFFERS_FOR_ESTIMATE
--- -------------------- ---------- ----------------- --------------------
  3 DEFAULT                   8192                24                 2976
  3 DEFAULT                   8192                48                 5952
  3 DEFAULT                   8192                72                 8928
  3 DEFAULT                   8192                96                11904
  3 DEFAULT                   8192               120                14880
  3 DEFAULT                   8192               144                17856
  3 DEFAULT                   8192               168                20832
  3 DEFAULT                   8192               192                23808
  3 DEFAULT                   8192               216                26784
  3 DEFAULT                   8192               240                29760
  3 DEFAULT                   8192               264                32736

 ID NAME             BLOCK SIZE SIZE FOR ESTIMATE BUFFERS FOR ESTIMATE
--- -------------------- ---------- ----------------- --------------------
  3 DEFAULT                   8192               276                34224
  3 DEFAULT                   8192               288                35712
  3 DEFAULT                   8192               312                38688
  3 DEFAULT                   8192               336                41664
  3 DEFAULT                   8192               360                44640
  3 DEFAULT                   8192               384                47616
  3 DEFAULT                   8192               408                50592
  3 DEFAULT                   8192               432                53568
  3 DEFAULT                   8192               456                56544
  3 DEFAULT                   8192               480                59520

已选择 21 行。
```

4.4.3　重做日志高速缓冲区（Redo buffer cache）

当用户执行了如 INSERT、UPDATE、DELETE、CREATE、ALTER 或 DROP 操作后，数据发生了变化，这些变化了的数据在写入数据库高速缓存之前会先写入重做日志缓冲区，同时变化之前的数据也放入重做日志高速缓存，这样在数据恢复时 Oracle 就知道哪些需要前滚哪些需要后滚了。

重做日志缓冲区的大小是可动态调节的，即在数据库运行期间修改这块内存的大小，Oracle

提供了一个初始化参数 LOG_BUFFER，在数据库实例启动时就分配好重做日志缓冲区的尺寸。例子 4-12 演示了如何查看重做日志缓存区。

例子 4-12　查看重做日志缓存区

```
SQL> show parameter log buffer;

NAME                                TYPE        VALUE
------------------------------------------------------------------------
log_buffer                                   integer     5611520
```



重做日志缓存区参数 log_buffer 是静态参数，不能动态修改，如果尝试修改会提示如下错误。

```
SQL> alter system set log_buffer = 1M;
alter system set log buffer = 1M
                     *
ERROR 位于第 1 行:
ORA-02095: 无法修改指定的初始化参数
```

4.4.4　大池（Large pool）和 Java 池（Java pool）

大池是 SGA 的一段可选内存区，只在共享服务器环境中配置大池。在共享服务器环境下，Oracle 在共享池中分配额外的空间用于存储用户进程和服务器进程之间的会话信息，但是用户进程区域 UGA（可理解为 PGA 在共享服务器中的另一个称呼）的大部分将在大池中分配，这样就减轻了共享池的负担。在大规模输入、输出及备份过程中也需要大池作为缓存空间。

Oracle 提高了参数 large_pool_size 参数设置大池的尺寸。先用例子 4-13 查看大池的尺寸。

例子 4-13　查看大池大小

```
SQL> show parameter large pool size

NAME                                TYPE        VALUE
---------------------------------------------------------- ------------
large_pool_size                             big integer    52M
```

参数 large_pool_size 是动态参数，可以通过 alter system 指令修改该参数的值，语句格式为：

```
SQL>alter system set large_pool_size = 48M
```

Java 池也是可选的一段内存区，但是在安装完 Java 或者使用 Java 程序时则必须设置 Java 池，它用于编译 Java 语言编写的指令。Java 语言与 PL/SQL 语言在数据库中有相同的存储方式。Oracle 提供了参数 JAVA_POOL_SIZE 设置 Java 池的大小。使用例子 4-14 查看当前 Java 池的大小。

例子 4-14　查看 java 池的大小

```
SQL> show parameter java pool size;

NAME                                TYPE        VALUE
---------------------------------------------------------- -------------
java_pool_size                         big integer        0
```



在 Oracle 11g 中，Java 池大小由数据库服务器在 SGA 中自动分配，当然用户也可以使用 alter system 指令修改该参数的值，在例子 4-14 中参数 java_pool_size 的值为 0 说明该参数为自动管理。

4.4.5 流池（Streaming pool）

流池也称为流内存，它是 Oracle 流专用的内存池，流（stream）是 Oracle 数据库中的一个数据共享，其大小可以通过参数 stream_pool_size 动态调整。

4.4.6 PGA（进程全局区）和 UGA（用户全局区）

进程全局区（PGA）是服务器进程专用的一块内存，它是操作系统进程专用的内存，系统中的其它进程是无法访问这块内存的。PGA 独立于 SGA，PGA 不会在 SGA 中出现，它是由操作系统在本地分配的。

1. PGA（进程全局区）

PGA 中存储了服务器进程或单独的后台进程的数据信息和控制信息。它随着服务器进程的创建而被分配内存，随着进程的终止而释放内存。PGA 与 SGA 不同，它不是一个共享区域，而是服务器进程专有的区域。在专有服务器（与共享服务器相对的概念）配置中包括如下的组件：

- 排序区：对某些的 SQL 语句执行结果进行排序。
- 会话信息：包含本次会话的用户权限和性能统计信息。
- 游标状态：标明当前会话执行的 SQL 语句的处理阶段。
- 堆栈区：包含其它的会话变量。



在共享服务器配置中，多个用户进程共享一个服务器进程，上述的一些内存区可能在 SGA 中分配。如果创建了大池，这些内存结构就存储在大池中，否则它们存储在共享池中。

图 4-6 和图 4-7 分别是专有服务器模式和共享服务器模式下的 PGA 结构图。

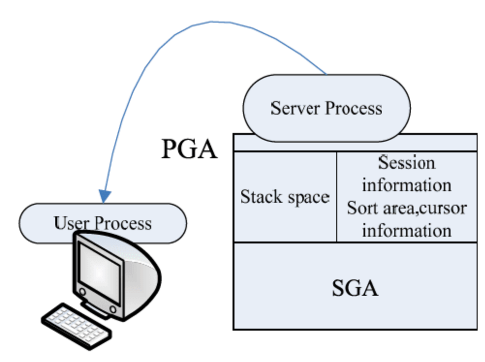


图 4-6　专有服务器模式下的 PGA 结构

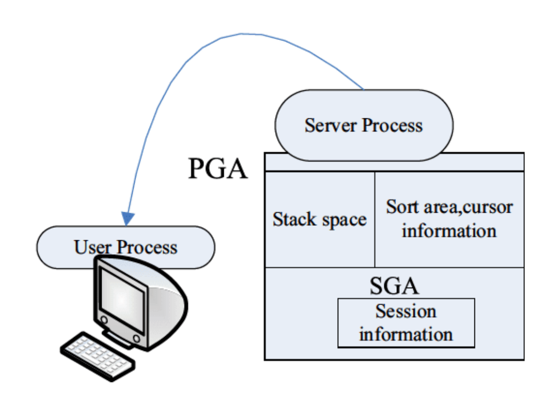


图 4-7　共享服务器模式下的 PGA 结构



在共享服务器结构中，会话信息是存储在 SGA 中的，两种模式下堆栈区（Stack space）都存储在 PGA 中。

2. UGA（用户全局区）

在共享服务器模式下有一个重要的概念即 UGA（用户全局区），它是用户的会话状态，这部分内存会话总可以访问，UGA 存储在每个共享服务器都可以访问的 SGA 中，这样任何服务器都可以使用用户会话的数据和其它信息。而在专有服务器模式下，用户会话状态不需要共享，用户进程与服务器进程是一一对应的关系，所以 UGA 总是在 PGA 中进行分配。

3. PGA 内存管理

从 Oracle 9i 开始，Oracle 提高了两种办法管理 PGA，即手动 PGA 管理和自动 PGA 管理。采用手动管理时，必须告诉 Oracle 一个特定的进程需要的排序区，允许使用多少内存，而在自动 PGA 管理中，则要求高速 Oracle 在系统范围内可以为 PGA 中的特定功能如排序区分配多少内存。例子 4-15 显示了笔者计算机上 PGA 中排序区的大小。

例子 4-15　查询 PGA 中排序区的大小

```
SQL> show parameter sort_area_size;

NAME                                  TYPE         VALUE
----------------------------- ------------------- ----------------
sort_area_size                                   integer       65536
```

在服务器进程最初查询时，会使用 512KB 内存实现数据排序，在 Oracle 将排序数据处理完之前，数据排序区的大小就由参数 SORT_AREA_SIZE 决定。



在 Oralce 10g 和 Oracle 11g 中可以实现共享服务器连接时 PGA 的自动管理，而在 Oracle 9i 中，使用共享服务器连接时只能使用手动 PGA 管理。

4.4.7 如何获得内存缓冲区的信息

SGA 是 Oracle 中所有进程共享的一段内存区，其中共享了数据库信息如数据库高速缓冲区中的数据、共享池中的库高速缓存中的 SQL 语句等。了解这些内存缓冲区的大小有助于理解 Oracle 的内存分配情况。例子 4-16 查看数据库的 SGA（系统全局区）。

例子 4-16 查看 SGA 中内存的分配情况

```
SQL> show sga;

Total System Global Area  535662592 bytes
Fixed Size                  1334380 bytes
Variable Size             260047764 bytes
Database Buffers          268435456 bytes
Redo Buffers                5844992 bytes
```

在上述输出中可以看到，SGA、Database Buffers 和 Redo Buffers 的大小，前面已经讲解了这些内存组件的作用。读者或许注意到了 Fixed Size 和 Variable Size 两个参数，它们和两个内存区有关，下面解释这两个内存区：

- 固定 SGA（和 fixed size 相关）：在固定 SGA 中，存储一组指向 SGA 中其他组件的变量。它的大小用户无法控制，因平台不同而有差异。但通常固定 SGA 区很小。Oracle 使用这个内存区来寻找其他 SGA 区，可以理解为数据库的自举区。
- 和 Variable Size 相关的内存区。该部分内存区包括共享池、Java 池和大池，其中 Variable Size 的大小要高于上述 3 个内存结构之和，因为在 Total SGA 中除去 db_cache_size 部分也包括在 Variable Size 中。

同时读者也可以使用例子 4-17 查询当前数据库的 SGA 尺寸。

例子 4-17 查看 SGA 的大小

```
SQL> show parameter sga max size;

NAME                                 TYPE        VALUE
------------------------------------ ------------------
sga_max_size                         big integer     512M
```



在 Oracle 11g 中该参数的值得到修正而使用 MB 作为单位，更利于识别，而在 Oracle 10g 版本中参数 sga_max_size 的值使用字节为单位。

4.5 Oracle服务器进程和用户进程

服务器进程和用户进程，是用户使用数据库连接工具同数据库服务器建立连接时，涉及的两个概念。

- 服务器进程：服务器进程犹如一个中介，完成用户的各种数据服务请求，而把数据库服务

器返回的数据和结果发给用户端。在专有连接中，一个服务器进程对应一个用户进程，二者是一一对应的关系。当用户连接中断，则服务器程序退出；在共享连接中，一个服务器进程对应几个用户进程，此时服务器进程通过 OPI（Oracle Program Interface）与数据库服务器通信。

- 用户进程：当用户使用数据库工具如 SQL*Plus 与数据库服务器建立连接时，就启动了一个用户进程，即 SQL*Plus 软件进程。例子 4-18 是使用 SQL*Plus 与数据库服务器成功建立连接的示例。

例子 4-18　使用 SCOTT 用户连接数据库

```
SQL> conn scott/tiger@orcl
已连接。
```

此时，用户和数据库服务器建立了连接，数据库服务器产生一个服务器进程，负责与数据库服务器的直接交互。

4.6　Oracle数据库后台进程

后台进程是在实例启动时，在数据库服务器端启动的管理程序，它使数据库的内存结构和数据库物理结构之间协调工作。从功能上考虑，在数据库内存结构、后台进程和数据库物理结构之间的关系如图 4-8 所示。

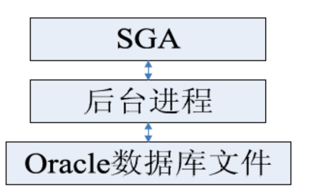


图 4-8　内存结构、后台进程和物理结构的关系图

数据库后台进程有 5 个是必须启动的，否则数据库实例无法启动成功。它们是 DBWR、LGWR、PMON、SMON 和 CKPT。本节主要讲解这 5 个后台进程，它们也是数据库服务器中最重要的几个后台进程。

4.6.1　系统监控进程（SMON）

系统监控进程的主要作用就是数据库实例恢复。当数据库发生故障时，如操作系统重启，此时实例 SGA 中的所有没有写到磁盘的信息都将丢失。当数据库重新启动后，系统监控进程自动恢复实例。实例恢复包括如下 3 个步骤。

- 前滚所有没有写入数据文件而记录在重做日志文件中的数据。此时，系统监控进程读取重做日志文件，把用户更改的数据重新写入数据块。
- 打开数据库，此时或许系统监控进程的前滚操作还没有完成，Oracle 这样做的目的就是方便用户及时登录，以免前滚时间太长，影响用户的行为，这样用户就可以操作那些没有被

事务恢复锁住的数据。

- 回滚未提交的事物。

除此之外，系统监控进程执行某些空间维护的作用。

- combine,coalesces,adjacent 数据文件中的自由空间。
- 回收数据文件中的临时段。

4.6.2 进程监控进程（PMON）

进程监控负责服务器进程的管理和维护工作，在进程失败或连接异常发生时该进程负责一些清理工作。

- 回滚没有提交的事务。
- 释放所持有的当前的表或行锁。
- 释放进程占用的 SGA 资源。
- 监视其它 Oracle 的后台进程，在必要时重启这些后台进程。
- 向 OracleTNS 监听器注册刚启动的实例。如果监听器在运行，就与这个监听器通信并传递如服务名和实例的负载等参数；如果监听器没有启动，进程监控（PMON）会定期地尝试连接监听器来注册实例。

4.6.3 数据库写进程（DBWR）

在介绍高速缓冲区时，提到了脏数据的概念，脏数据就是用户更改了的但没有提交的数据库中的数据，因为在数据库的数据文件与数据库高速缓存中的数据不一致，故称为脏数据，这种脏数据必须在特定的条件下写到数据文件中，这就是数据库写进程的作用。

数据库写进程负责把数据库高速缓冲区中的脏数据写到数据文件中。或许读者会问，为什么不立即提交脏数据呢，这样就不需要复杂的数据库写进程来管理。其实，Oracle 这样设计的思路很简单，就是减少 I/O 次数，但脏数据量达到一定程度或者某种其它条件满足时，就提交一次脏数据。因为磁盘的输入、输出会花费系统时间，使得 Oracle 系统的效率不高。

图 4-9 是数据库写进程涉及的数据库组件，通过该图可以清晰地理解数据库写进程工作中涉及的“实体”。

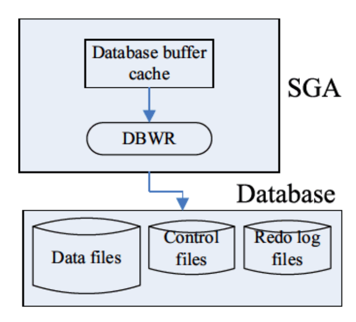


图 4-9 数据库写进程“实体”关系

当一个事件发生时，会触发数据库写进程把脏数据写到数据库的数据文件中。

- 发生检查点事件。
- 脏数据量达到了门限值。
- 数据库缓冲区没有足够的缓存为其它事务提供足够的空间。
- 表空间处于热备份状态。
- 表空间被置为离线状态。
- 表空间被置为只读状态。
- 删除表或者截断表。
- 超时。



数据库写进程的性能显然很重要，如果它写脏数据到数据文件的速度很慢，使大量缓冲区无法释放，就会出现一些等待事件，如 Free Buffer Waits 等。实际在 Oracle 数据库上，一个数据库实例可以启动多个数据库写进程，在多 CPU 系统可以使用多个数据库写进程来分担单个写进程的工作负载。

4.6.4　重做日志写进程（LGWR）

重做日志写进程负责将重做日志缓冲区中的数据写到重做日志文件。此时重做日志缓冲区中的内容是恢复事务所需要的信息，比如用户使用 UPDATE 语句更新了某行数据，恢复事务所需的信息就是更新前的数据和更新后的数据，这些信息用于该事务的恢复。

重做日志写进程在满足一个条件时，会启动进程工作。

- 当事务提交时。
- 当重做日志缓冲区的 1/3 被占用时。
- 当重做日志缓冲区中有 1MB 的数据时。
- 当数据库写进程把脏数据写到数据文件之前。

图 4-10 是重做日志写进程工作示例图。

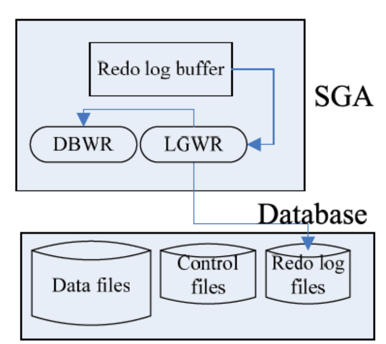


图 4-10　数据库重做日志写进程



从图 4-10 可以看出，日志写进程会通知数据库写进程将脏数据写到数据文件，但是数据库写进程不会把脏数据写到在线重做日志，也不会通知日志写进程做任何事情。

数据库写进程是离散写到不同数据库文件上的，在执行一个更新时，数据库写进程会修改不同空间中存储的数据块和索引块，所以数据库写进程的离散写的速度很慢。而重做日志写进程是顺序写，它比离散写的效率要高，把每个事务的重做信息全部放在重做日志中。通过在数据库高速缓存中缓存脏数据块，而由重做日志写进程完成大规模顺序写，从整体上可以提高系统的性能。

4.6.5 归档日志进程（ARCH）

归档日志进程是可选进程，该进程并不在实例启动时自动启动。它的作用是把写满的重做日志文件的数据写到一个归档日志中，这个归档日志用作介质故障时的数据库恢复。图 4-11 是归档日志进程涉及的实体关系，即归档日志和重做日志之间的关系。

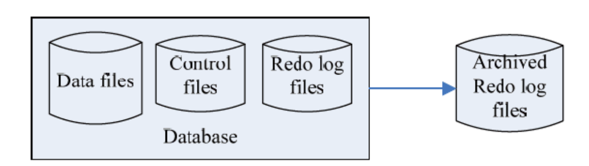


图 4-11 归档日志进程

重做日志文件负载实例失败时的数据恢复，因为 SGA 中没有被保存的数据会全部丢失，这样使用重做日志文件就可以完全恢复事务。而归档日志进程用于介质恢复，比如磁盘损坏，可以使用以前备份的数据文件，使用归档日志和重做日志就可以完全恢复数据库。

归档进程不在实例启动时自动启动，在生产数据库中必须使用归档模式，以防止灾难性的数据损坏。可以使用例子 4-19 查看系统的归档模式。

例子 4-19 查看系统的归档模式

```
SQL> connect system/oracle@orcl as sysdba;
已连接。
SQL> archive log list;
数据库日志模式              非存档模式
自动存档              禁用
存档终点              USE DB RECOVERY FILE DEST
最早的联机日志序列        3
当前日志序列              5
```

例子 4-19 说明当前的数据库处于非归档模式，没有启动归档进程，归档文件存储在 USE_DB_RECOVERY_FILE_DEST 参数指定的目录下。那么，如何设置数据库为归档模式呢？

可以使用例子 4-20 设置数据库为归档模式，并且启用自动归档，但前提是必须首先关闭数据库，以 MOUNT 参数启动数据库，更改后再启动到 OPEN 状态即可。下面演示这个过程，注意这里在迁移时已经关闭了数据库。

例子 4-20　设置数据库为归档模式的过程

```
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> connect /as sysdba;
已连接到空闲例程。
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area  535662592 bytes
Fixed Size                  1334380 bytes
Variable Size             260047764 bytes
Database Buffers          268435456 bytes
Redo Buffers                5844992 bytes
数据库装载完毕。
SQL> alter database archivelog;

数据库已更改。

SQL> alter database open;

数据库已更改。
```

再使用例子 4-21 查询更改结果。

例子 4-21　查询当前数据库的归档模式

```
SQL> archive log list;
数据库日志模式              存档模式
自动存档                启用
存档终点               USE_DB_RECOVERY_FILE_DEST
最早的联机日志序列        7
下一个存档日志序列      9
当前日志序列              9
```

在例子 4-21 中，数据库的日志模式改为存档模式，自动存档已经启动，归档日志的存储目录即存档终点是数据库恢复文件目录。

我们可以通过例子 4-22 查询参数 DB_RECOVERY_FILE_DEST 的位置。

例子 4-22　查看数据库恢复目录的位置

```
SQL> show parameter db_recovery

NAME                         TYPE        VALUE
------------------------------------------------------------------------
db_recovery_file_dest       string        F:\app\oracle\flash_recovery_area
db_recovery_file_dest_size         big integer     3852M
```

参数 db_recovery_file_dest 的 value 为 F:\app\Administrator\flash_recovery_area，说明了数据库恢复文件目录的位置，而参数 db_recovery_file_dest_size 的 value 值为 2GB 说明该存储目录分配的磁盘空间大小。

4.6.6 校验点进程（Checkpoint process）

首先介绍检验点，检验点是一个事件，当数据库写进程把 SGA 中所有被修改了的数据库高速缓冲中的数据写到数据文件上时产生，这些被修改的数据包括提交的和未提交的数据。由于引入了校验点，使得所有的校验点的所有变化了的数据都写到数据文件中，在实例恢复时，就不必恢复校验点之前的重做日志中的数据，加快了系统恢复的效率。

校验点进程并不是用于建立校验点，只是在校验点发生时，会触发这个进程进行一系列工作。包括如下几点：

- 校验点进程要将校验点号码写入相关的数据文件的文件头中。
- 校验点进程把校验点号码、SCN 号、重做日志序列号、归档日志名字等都写入控制文件。

图 4-12 是数据库校验点进程的工作示意图。

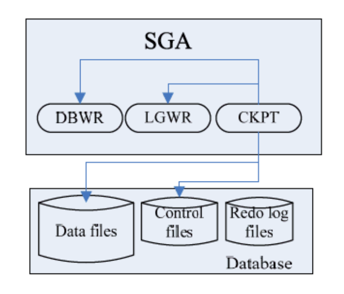


图 4-12　数据库校验点进程

Oracle 提供一个指令，用户可以强制产生校验点，使得用户可以干预校验点的产生，如例子 4-23 所示。

例子 4-23　强制执行校验点

```
SQL> alter system checkpoint;
```

4.7　本章小结

本节首先给出 Oracle 数据库体系结构的整体框图，读者通过此结构框图可以清晰地了解各个组件。接下来把体系结构分成数据库内存结构、数据库文件结构和数据库后台进程，逐个讲解了各种结构的包含组件及其相应的功能。读者应该重点理解内存结构和后台进程，内存结构为数据库的工作分配了动态的区域，后台进程协调、监控数据库内存结构和物理文件之间的关系，使得数据库系统协调地工作，完成数据的维护和管理任务。物理结构相对简单，包括 3 个文件即数据文件、控制文件和重做日志文件。数据文件保存用户数据，控制文件保存了各种启动数据库所需的信息，日志文件主要用于数据库恢复。

数据库实例是很重要的概念，实例指一组内存结构和后台进程，而数据库服务器包括实例和数据库，我们经常说的启动数据库服务器实际上是先启动数据库实例，而后挂接数据库。用户进程可以通过数据库服务器的服务器进程操作数据库。

第 5 章

SQL语言概述

SQL 语言是“结构化查询语言”的意思，即 Structured Query Language。两个工业界认可的国际机构 ANSI 和 ISO 把 SQL 作为关系数据库的标准语言。SQL 语言涉及的语句简单，语义明了，如果读者懂些英文，则很容易掌握 SQL 语言。使用该语言检索和维护数据库，编写涉及数据库操作的应用程序或脚本语言。在实际工作中，我们经常使用数据查询语句和数据操纵语句。在使用这些 SQL 语句时，可以使用一些函数来处理输出结果或者通过分组函数使得输出数据更加友好。本章将依次介绍这些内容。

5.1 SQL语句分类

SQL 语句按照其功能分为 5 类，即数据查询语句、数据操纵语句、数据定义语句、事务控制语句和数据控制语句。下面通过 SQL 语句关键字依次简单介绍这些语句的功能，在本书后续的章节中读者会学到如何使用这些语句，以及使用这些语句的场合。

（1）数据查询语句

SELECT：该语句的功能是从数据库中获得用户数据，如查询一个表中的全部数据等。

（2）数据操纵语句（DML）

- INSERT：该语句的功能是向表中添加记录。
- UPDATE：该语句的功能是更新表中的数据，通常和 WHERE 条件语句一起使用。
- DELETE：删除表中的数据。

（3）数据定义语句（DDL）

- CREATE：创建数据库对象如表、索引、视图等。
- ALTER：改变系统参数，如改变 SGA 的大小等。
- DROP：删除掉一个对象，如删除一个表、索引或者序列号等。
- RENAME：重命名一个对象。
- TRUNCATE：截断一个表。

（5）事务控制语句

- COMMIT：用于提交由 DML 语句操作的事务。

- ROLLBACK：用于回滚 DML 语句改变了的数据。

（6）数据控制语句

- GRANT：用于授予用户访问某对象的特权。
- REVOKE：用于回收用户访问某对象的特权。

读者只需要理解对 SQL 语句存在这些分类和每种 SQL 语句在 Oracle 数据库操作中的作用，本章将重点介绍数据查询语句和数据操纵语句。

5.2 SQL的查询语句

Oracle 的 SQL 查询语句即 SELECT 语句。如果需要检索数据库中的数据，就需要使用该语句。在使用 SELECT 语句时，必须有相应的 FROM 子句。当需要复杂查询时可以使用 WHERE 子句。把整个查询语句中的 SELECT、FROM 和 WHERE 称为关键字，下面详细介绍查询语句的用法和各个关键字的含义。

5.2.1 SELECT 语句的语法及书写要求

一个简单的 SELECT 语句至少包含一个 SELECT 子句和一个 FROM 子句。其中 SELECT 子句指明要显示的列，而 FROM 子句指明包含要查询的表，该表包含了在 SELECT 子句中的列。其语法格式化如下：

```
SELECT *|{[DISTINCT] column | expression [alias],……}
FROM    table;
```

说明在上述语法规则中，|号表示或的关系，[]表示可选。

其中：

```
SELECT              选择一个列或多个列。
*                   选择表中所有的列。
DISTINCT            去掉列中重复的值。
column|expression   选择列的名字或表达式。
alias               为指定的列设置不同的标题。
FROM table          指定要选择的列所在的表，即对那个表进行数据检索。
```

上面涉及的语法，在下面都会介绍。其中有几个术语需要读者分辨清楚，因为在接下来的内容中将多次用到，它们是关键字、子句和语句。其区别如下：

- 关键字：它是一个单独的 SQL 元素，如 SELECT、FROM 等都是关键字，并且要求关键字不能简写，如写成 SEL、FRO 是不允许的，但是不要求必须大写，这里采用大写是采用 Oracle 推荐的写法，即关键字都大写而其他小写以做区分。
- 子句：子句是一个 SQL 语句的一部分，它不是一个可执行的 SQL 语句。如 SELECT * 就是一个子句。
- 语句：语句由一个或多个子句组成，它是可执行的。如 SELECT * FROM dept 就是一个语句。在书写语句时，读者最好采取每个子句一行的习惯，这样可增强可读性。

5.2.2　简单查询

先使用一个例子说明如何实现一个简单的查询，此时我们使用用户 SCOTT（该用户在创建数据库时会自动创建）的 dept 表，如例子 5-1 所示。

例子 5-1　查询 SCOTT 用户的 dept 表的全部内容

```
SQL> conn scott/tiger
已连接。
SQL> SELECT *
  2  FROM dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        50 OPERATIONS     BOSTON
```

首先使用 SCOTT 用户登录，该用户的默认密码是 TIGER，此时不区分用户名和密码的大小写。然后输入一个查询语句，该语句的作用是查询表 dept 中的所有列的数据。

这里*号的含义是选择表中的所有列，FROM 关键字后是表名。表 dept 是一个部门表，该表有三列，分别是 DEPTNO（部门号）、DNAME（部门名称）和 LOC（部门所在地）。

还有一种查询方式实现查询表中的所有列的数据，即在 SELECT 关键字后输入所有列的名字，名字之间用逗号分开。如例子 5-2 中，我们重新查询表 dept 中的所有列的数据。

例子 5-2　重新查询表 dept 的全部内容

```
SQL> SELECT deptno,dname,loc
  2  FROM dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        50 OPERATIONS     BOSTON
```

在例子 5-2 中，SELECT 关键字之后的列名用逗号分开。



例子 5-1 和例子 5-2 都是操作用户 SCOTT 的表，如果要操作顺利需要读者使用 SCOTT 用户登录，如果使用 SYSTEM 用户登录，就会提示错误，我们看例子 5-3。

例子 5-3　使用 SYSTEM 用户登录，该用户的默认密码是 MANAGER

```
SQL> conn system/manager
已连接。
SQL> SELECT *
  2  FROM dept;
FROM dept
```

```
     *
ERROR 位于第 2 行:
ORA-00952: 表或视图不存在
```

此时如果在 FROM 子句改为 FROM scott.dept，该语句就会顺利执行，如例子 5-4 所示。

例子 5-4　在 SYSTEM 用户模式下使用“模式名.表名”的方式查询表数据

```
SQL> conn system/manager
已连接。
SQL> SELECT *
  2  FROM scott.dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        50 OPERATIONS     BOSTON
```

因为 SYSTEM 用户为系统管理员用户，所以他有权限操作 SCOTT 用户的对象，使用 SYSTEM 用户登录，在查询用户 SCOTT 的对象时需要只要在对象前指明是该用户的对象就可以。

5.2.3　特定的列查询

在实际中，并不是表中所有的列都需要查询，这样用户只需要查询所需要的列，此时只需要在 SELECT 关键字后输入要查询的列名，如例子 5-5 所示。

例子 5-5　查询表 dept 的特定列的数据

```
SQL> SELECT dname,loc
  2  FROM dept;

DNAME          LOC
-------------- -------------
ACCOUNTING     NEW YORK
RESEARCH       DALLAS
SALES          CHICAGO
OPERATIONS     BOSTON
```

其实，在 SELECT 之后可以输入表中存在的任意的列，并且列的顺序没有要求，数据的显示将以用户输入的列的顺序为基准，如例子 5-6 所示。

例子 5-6　查询表 dept 中的任意列的数据

```
SQL> SELECT dname,loc,deptno
  2  FROM dept;

DNAME          LOC               DEPTNO
-------------- ------------- ----------
ACCOUNTING     NEW YORK              10
RESEARCH       DALLAS                20
SALES          CHICAGO               30
```

```
OPERATIONS       BOSTON                   50
```

5.2.4　WHERE 子句

对于一个特定的表，如果用户想查询一个特定条件的表该怎么办呢。Oracle 提供了 WHERE 子句来限制查询条件，WHERE 子句可以限制选择的行数。这样可以实现满足一定条件的数据查询，实现更加灵活的应用。如例子 5-7 查询 SCOTT 用户的表 dept 中满足部门驻地在 CHICAGO 的部门所有信息。

例子 5-7　使用 WHERE 子句查询 SCOTT 用户的表 dept 的全部数据

```
SQL> SELECT *
  2  FROM dept
  3  WHERE loc = 'CHICAGO';

    DEPTNO  DNAME          LOC
----------------- --------------------- -------------
        30  SALES          CHICAGO
```

上述查询虽然使用了 SELECT *子句，但是 WHERE 子句限制了查询的结果必须是 loc 为 CHICAGO 的部门信息，所以限制了查询的行数。当然，用户也可以输入其他限制性条件，如查询部门号小于 30 的部门信息，如例子 5-8 所示。

例子 5-8　查询表 dept 中部门号小于 30 所有数据

```
SQL> SELECT *
  2  FROM dept
  3  WHERE deptno < 30;

    DEPTNO  DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
```

WHERE 子句中的条件可以根据需要通过各种算数或逻辑运算符实现条件限制。

5.2.5　列标题的默认显示格式

在上面几节介绍的查询语句中，显示的结果是 Oracle 提供的，我们并没有对显示的信息做任何修改，即数据的显示结果是 Oracle 的默认结果。在 Oralce 中列标题的显示满足如下规则。

- 字符和日期型的列标题靠显示宽度的左边。
- 数字型的列标题靠显示宽度的右边。
- 默认的列标题都是大写，如图 5-1 所示。

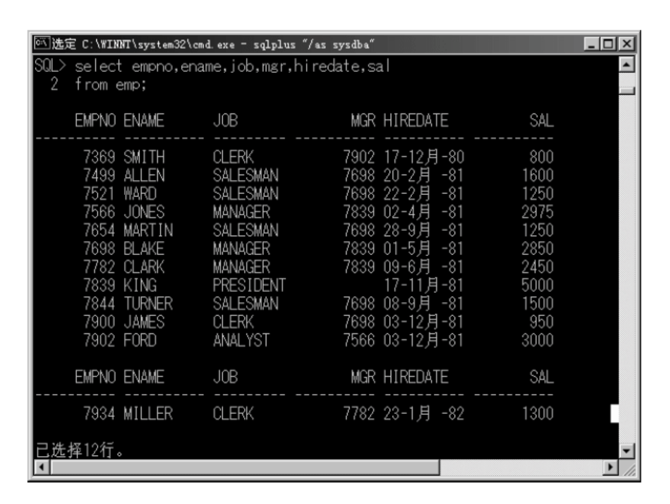


图 5-1　列标题的默认属性

5.2.6　在 SQL 语句中使用列的别名

在使用 SELECT 语句时，SQL*Plus 使用选择的列名作为列标题，并且采用大写方式。由于表中的列名是数据库开发人员或程序员设计的，是为了编程的需要。但是这样的列标题可能不具备描述性而难以理解，Oracle 提供了列别名更改列标题的显示方式，先看例子 5-9。

例子 5-9　通过别名更改列标题的查询

```
SQL>SELECT empno,ename employee_name,sal AS salary,deptno "Deptmentnumber"
  2* FROM emp

 EMPNO EMPLOYEE NAME          SALARY Deptmentnumber
-------------- ---------------------------------------- ----------------
     7369 SMITH                  800            20
     7599 ALLEN                 1600            30
     7521 WARD                  1250            30
     7566 JONES                  2975            20
     7655 MARTIN                1250            30
     7698 BLAKE                  2850            30
     7782 CLARK                 2550            10
     7839 KING                   5000            10
     7855 TURNER                1500            30
     7900 JAMES                   950            30
     7902 FORD                  3000            20

  EMPNO EMPLOYEE NAME          SALARY Deptmentnumber
-------------- ------------------------------------- --------------------
     7935 MILLER                1300            10

已选择 12 行。
```

创建别名时，在列名后使用 AS 关键字，之后紧跟别名，或者在列名后加空格，然后紧接着别名，如上例中 ename employee_name 和 sal AS salary，但是此时的列标题在显示时为别名的大写格

式。如果要保持别名的格式，可以使用双引号如 deptno "Deptmentnumber"。

当然，也可以使用中文名字，如 5.2.7 节例子 5-10 中所示。

5.2.7　算数运算符及使用

算数运算符即加减乘除四种运算：+、-、*、/。使用算数运算符实现对日期型和数字型列的算数操作。创建一个具有算数运算的表达式，丰富查询的显示结果。

如在 SCOTT 用户的 EMP 表中，查询每个员工的年薪。使用例子 5-10 查询员工的名字和年薪。

例子 5-10　查询 SCOTT 用户 EMP 表中员工的名字和年薪

```
SQL> SELECT ename "员工姓名",sal*12 "年薪"
  2  FROM emp
  3  WHERE job = 'MANAGER';

员工姓名             年薪
-------------------- ----------
JONES                35700
BLAKE                35200
CLARK                29500
```

其他运算符的使用规则类似，读者可以自行测试，如为所有 job= “SALESMAN”的员工月薪增加 500。

算数运算符遵循一定的优先顺序，即“乘除”优先于“加减”，而“乘除”具有同等优先权，“加减”也具有同等优先权。同等优先权的运算符按照从左到右的顺序计算。

如 sal*12 + 1000，先计算 sal*12，再加 1000 就是该表达式的最后计算结果，如例子 5-11 所示。

例子 5-11　在查询中使用运算符

```
SQL>SELECT ename "员工姓名",sal*12+1000 "年薪"
  2  FROM emp
  3* WHERE job = 'MANAGER'

员工姓名             年薪
---------- ----------
JONES                36700
BLAKE                35200
CLARK                30500
```

5.2.8　DISTINCT 运算符

DISTINCT 运算符使得查询的结果没有重复内容，如需要查询 SCOTT 用户的 EMP 表中有多少个 job。我们先用例子 5-12 测试不使用 DISTINCT 的查询结果，再使用例子 5-13 测试使用 DISTINCT 的查询结果，两个结果对比，读者可以清晰体会使用 DISTINCT 的区别。

例子 5-12　查询表 EMP 中的 job 名

```
SQL> SELECT job
  2  FROM emp;
```

```
JOB
---------
CLERK
SALESMAN
SALESMAN
MANAGER
SALESMAN
MANAGER
MANAGER
PRESIDENT
SALESMAN
CLERK
ANALYST

JOB
---------
CLERK

已选择 12 行。
```

例子 5-12 中，选择结果有 12 行，重复的 job 内容也显示在结果中，显然这并不是我们想要的结果，而例子 5-13 使用 DISTINCT 关键字实现不重复查询。

例子 5-13　使用 DISTINCT 关键字实现不重复查询表 emp 中的 job 名

```
SQL> SELECT distinct job
  2  FROM emp;

JOB
---------
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN
```

在 SELECT 关键字紧跟 DISTINCT 关键字，使得选择的行没有重复的结果，但是如果 DISTINCT 关键字后有多个列，情况如何呢？如例子 5-14 所示。

例子 5-14　使用 DISTINCT 关键字实现多列查询

```
SQL> SELECT distinct deptno,job
  2  FROM emp;

  DEPTNO JOB
--------------- ----------------
       10 CLERK
       10 MANAGER
       10 PRESIDENT
       20 ANALYST
       20 CLERK
       20 MANAGER
```

```
          30 CLERK
          30 MANAGER
          30 SALESMAN

    已选择 9 行。
```

此时使用 DISTINCT 关键字使得结果中多个列的组合没有重复的结果，即每一行数据不完全相同。

5.2.9　连接运算符及使用

连接运算符把列与其他列连接起来，也可以把列与字符串连接起来。连接符是两个竖线“||”，在连接字符串时使用单引号。例子 5-15 是使用连接运算符的例子。

例子 5-15　使用连接运算符“||”

```
    SQL> SELECT ename  ||' is a '||job ||' and 1month salary is:'|| sal  As "The
implpyees's information"
      2  FROM emp;

    The imployees's information
    -----------------------------------------------------------------------------
    SMITH is a CLERK and 1month salary is:800
    ALLEN is a SALESMAN and 1month salary is:1600
    WARD is a SALESMAN and 1month salary is:1250
    JONES is a MANAGER and 1month salary is:2975
    MARTIN is a SALESMAN and 1month salary is:1250
    BLAKE is a MANAGER and 1month salary is:2850
    CLARK is a MANAGER and 1month salary is:2550
    KING is a PRESIDENT and 1month salary is:5000
    TURNER is a SALESMAN and 1month salary is:1500
    JAMES is a CLERK and 1month salary is:950
    FORD is a ANALYST and 1month salary is:3000

    The imployees's information
    ------------------------------------------------------------------------------
    MILLER is a CLERK and 1month salary is:1300

    已选择 12 行。
```

该例子中使用了 4 个连接运算符，把三列 ename、job、sal 和两个字符串（‘is a’和‘and 1 month salary is:’）连接起来。显然这样的显示结果更容易阅读。在上例中，我们也使用了别名，即将显示的信息设置一个列标题为“The imployees's information”。

5.3　书写规范

在书写 SQL 语句时，不区分大小写，如 SELECT 和 SELECT 都是允许的。但是关键字不能跨行书写，也不能缩写比如 SELECT 不能写成 SEL。一个 SQL 语句可以有多行

Oracle 推荐了书写 SQL 语句的规范，使用该规范会使得 SQL 语句更加容易阅读，并且容易区分 SQL 语句关键字和其它对象名。推荐的规范如下。

- SQL 语句的关键字要大写，对象名小写。
- 缩进对齐，这样便于阅读。
- 每个子句一行。

下面给出一个例子。

例子 5-16　查询表 EMP 中工资大于 1500 的员工信息

```
SQL> SELECT empno,ename,job,mgr,hiredate,sal
  2  FROM emp
  3  WHERE  sal > 1500;

    EMPNO ENAME      JOB              MGR HIREDATE          SAL
---------- ---------- --------- ---------- ---------- ----------
     7599 ALLEN      SALESMAN        7698 20-2 月 -81       1600
     7566 JONES      MANAGER         7839 02-5 月 -81       2975
     7698 BLAKE      MANAGER         7839 01-5 月 -81       2850
     7782 CLARK      MANAGER         7839 09-6 月 -81       2550
     7839 KING       PRESIDENT            17-11 月-81       5000
     7902 FORD       ANALYST         7566 03-12 月-81       3000

已选择 6 行。
```

每个子句一行使得阅读方便，通过 SQL 关键字大写，使得它们与 Oracle 对象区分开来，这样整个代码就很清晰。

5.4　单行函数

为了方便数据库的操作，Oracle 提供了各种函数操作，单行函数分为字符型单行函数、数字型单行函数和日期型单行函数。本节依次讲解这三类函数。

5.4.1　字符型单行函数

字符型单行函数接受一个字符输入，并且返回一个计算结果，该结果可以是字符型，也可以是数字型。常用的单行字符型函数如下。

LOWER，其函数格式为 LOWER(column | expression)，函数功能是把字符串转换成小写。如例子 5-17 所示。

例子 5-17　使用单行函数 LOWER（）

```
SQL> SELECT LOWER('Structured Query Language')
  2  FROM dual;

LOWER('STRUCTUREDQUERYLAN
```

```
-------------------------
structured query language
```

UPPER，其函数格式为 UPPER(column | expression) ，函数功能是把字符串转换成大写，如例子 5-18 所示。

例子 5-18　使用单行函数 UPPER（）

```
SQL> SELECT UPPER('Structured Query Language')
  2  FROM dual;

UPPER('STRUCTUREDQUERYLAN
-------------------------
STRUCTURED QUERY LANGUAGE
```

INITCAP，其函数格式为 INITCAP(column | expression)，其功能是把字符串的首字母大写，如例子 5-19 所示。

例子 5-19　使用单行函数 INITCAP（）

```
SQL> SELECT INITCAP('structured query language')
  2  FROM dual;

INITCAP('STRUCTUREDQUERYL
-------------------------
Structured Query Language
```

CONCAT，其函数格式为 CONCAT(column1 | expression1, Column2 | expression2)，该函数用于连接两个字符串，或者连接两个列中的数据。例子 5-20 连接两个字符串。

例子 5-20　使用单行函数 CONCAT（）

```
SQL> SELECT CONCAT ('Structured Query Language','is easy to learn!')
  2  FROM dual;

CONCAT('STRUCTUREDQUERYLANGUAGE','ISEASYTO
------------------------------------------
Structured Query Languageis easy to learn!
```

在函数 CONCAT 中，参数也可以是列名，列名和表达式可以根据需要自由选择，如例子 5-21 所示。

例子 5-21　使用列名和表达式的 CONCAT 函数

```
SQL> SELECT concat(ename,hiredate) "emp_name,hiredate"
  2  FROM emp;

emp name,hiredate
--------------------
SMITH17-12 月-80
ALLEN20-2 月 -81
WARD22-2 月 -81
JONES02-5 月 -81
MARTIN28-9 月 -81
```

```
BLAKE01-5 月 -81
CLARK09-6 月 -81
SCOTT19-5 月 -87
KING17-11 月-81
TURNER08-9 月 -81
ADAMS23-5 月 -87

emp name,hiredate
--------------------
JAMES03-12 月-81
FORD03-12 月-81
MILLER23-1 月 -82

已选择 15 行。
```

SUBSTR，其函数格式为 SUBSTR(column | expression,m [,n])，该函数从一个字符串中获取一个子串，该子串从 expression 的第 m 个字符开始，到第 n 个字符结束，如果不指定 n 则从 m 个字符开始到 expression 表达式的结尾，如例子 5-22 所示。

例子 5-22　使用 SUBSTR 函数

```
SQL> SELECT SUBSTR('structured query language',12)
  2  FROM dual;

SUBSTR('STRUCT
--------------
query language
```



函数 SUBSTR 在计算子串的起始位置时，一个空格占有一个字符。上述字符串'structured query language'共有 25 个字符，第 12 个字符是'q'，而起始参数 m = 12，没有指定结束字符的位置，所以默认从 12 个字符开始到字符串的结尾。

LENGTH，其函数格式为 LENGTH(column | expression)，计算字符串中的字符个数。例子 5-23 测试字符串'structured query language'中的字符数。

例子 5-23　使用 LENGTH 函数

```
SQL> SELECT LENGTH('structured query language')
  2  FROM dual;

LENGTH('STRUCTUREDQUERYLANGUAGE')
---------------------------------
                               25
```

INSTR，其函数格式为 INSTR(column | expression, 'string', [,m], [n])，该函数的功能是在字符串 expression 中搜索字符串'string'，参数 m 和 n 指定搜索的开始位置和结束位置。如果没有指定 m 和 n 的值，则从字符串 expression 中搜索，如例子 5-24 所示。

例子 5-24　使用 INSTR 函数

```
SQL> SELECT instr('structured query language','query')
```

```
  2  FROM dual;

INSTR('STRUCTUREDQUERYLANGUAGE','QUERY')
----------------------------------------
                                      12
```

该函数返回一个数字，表明字符串'query'在字符串'structured query language'中的起始位置。

LPAD|RPAD，其函数格式为 LPAD|RPAD (column | expression，n, ‘string’)，我们只给出一个例子 5-25，通过例子说明该函数的作用。

例子 5-25　使用 LPAD 函数

```
SQL> SELECT LPAD(sal,10,'*')
  2  FROM emp
  3  WHERE sal >1500;

LPAD(SAL,10,'*')
--------------------
******1600
******2975
******2850
******2550
******5000
******3000

已选择 6 行。
```

函数 LPAD(sal,10,'*')中，sal 是表 emp 中的列名，10 表示该函数的输出结果需要 10 个字符，而*号表示如果列 sal 的值不足 10 个字符，则在 sal 值的左边用*号补充。如果此时函数为 RPAD(sal,10,'*')，则在 sal 值的右边用*号补充。读者可以自己测试。函数 LPAD 和 RPAD 的作用就是在输出结果中增加一些补充信息，使得输出结果更具有可读性。

TRIM，其函数格式为 TRIM (leading|trailing|both, trim_character FROMTrim_source)，该函数的作用是在字符串中剪切一个字符，输出结果是一个字符串。其参数中 leading|trailing|both 的作用分别是函数从源字符串的头部删除要剪切的字符还是从尾部和两边删除要剪切的字符。默认是 both。

例子 5-26　使用 TRIM 函数

```
SQL>SELECT trim ('S' FROM 'SQL is an easy Database languageS')
  2  FROM dual;

TRIM('S'FROM'SQLISANEASYDATABAS
-------------------------------
QL is an easy Database language
```

输出结果显示已经把源字符串'SQL is an easy Database languageS'中的第一个 S 和最后一个 S 都已经删除掉。

REPLACE，其函数格式为：REPLACE (text,search_string,replacement_string)，该函数把源字符串（text）中的某个字符串（search_string）替换为另一个字符串（replacement_string）。该函数很

简单，给出一个例子供读者体会。

例子 5-27　使用 REPLACE 函数

```
    SQL> SELECT replace ('sql is an easy Database language','sql','Structured Query
Language')
      2  FROM dual;

    REPLACE('SQLISANEASYDATABASELANG
    --------------------------------
    Structured Query Language is an easy Database language
```

该函数将源字符串中的 sql 替换为 Structured Query Language，用完整的英语单词更具有说明性，容易理解。

5.4.2　数字型单行函数

数字型单行函数实现对数字的处理，其输出也是数字类型。它包括 3 个函数，如下所示。

- ROUND(column/expression , n)
- TRUNC(column/expression , n)
- MOD(m , n)

下面依次介绍这些函数的具体使用。

ROUND 函数。该函数的作用是对一个数字，输出用户指定的小数位，如数字 32.1515，用户可以要求只输出小数点后的 3 位，但是该函数处理数字时使用四舍五入的规则，如例子 5-28 所示。

例子 5-28　使用 ROUND 函数

```
SQL> SELECT round(32.1515,3)
  2  FROM dual;

ROUND(32.1515,3)
-------------------
           3.152
```

如果该函数参数 n 为负数，则表示要求保留相应的整数位，如例子 5-29 所示。

例子 5-29　ROUND 函数参数 n 为负数的取值

```
SQL> SELECT round(32.1515,-1)
  2  FROM dual;

ROUND(32.1515,-1)
-----------------
             30
```

TRUNC 函数。该函数的作用是截断一个数字，只保留小数点后一定的位数，该函数处理数字时不使用四舍五入规则，显然 Oracle 使用截断一词的用意也是如此。

例子 5-30　使用 TRUNC 函数

```
SQL> SELECT trunc (32.1515,3)
  2  FROM dual;

TRUNC(32.1515,3)
----------------
          32.151
```

例子 5-31　使用 TRUNC 函数

```
SQL> SELECT trunc (32.1515,-1)
  2  FROM dual;

TRUNC(32.1515,-1)
-----------------
               30
```

MOD 函数。该函数的作用是求余数，如例子 5-32 所示。

例子 5-32　使用 MOD 函数（够除）

```
SQL> SELECT mod(1000,500)
  2  FROM dual;

MOD(1000,500)
-------------
          200
```

该例子中用 1000 除以 500，商为 2，此时余数是 200，即 2×500＋200（余数）＝1000，所以经过计算之后的结果是 200。但是如果不够除又怎么办呢。我们用 100 除以 500 显然不够除，我们用例子 5-33 测试结果。

例子 5-33　使用 MOD 函数（不够除）

```
SQL> SELECT mod (100,500)
  2  FROM dual;

MOD(100,500)
------------
         100
```

显然 100/500 不够除，商为 0，此时余数是 100，即 0×500＋100（余数）=100。

5.4.3　日期型单行函数

Oracle 使用内部数字格式存储日期。默认的日期显示和输入格式为 DD-MON-RR。有效日期从公元前 5712 年 1 月 1 日到公元 9999 年 12 月 31 日。日期在数据库中的内部存储格式为：世纪、年、月、日、时、分、秒。不论外部的日期形式如何改变，数据库对日期的内部存储格式是不会变的。

Oracle 提供了用于操作或显示日期的函数，它们包括：SYSDATE，MONTHS_BETWEEN，ADD_MONTHS，NEXT_DAY，LAST_DAY。下面依次介绍这些函数。

SYSDATE 函数。该函数返回系统的当前日期，该日期受操作系统限制，即 Oracle 数据库读取操作系统的时间，如例子 5-34 所示。

例子 5-34　查询 SYSDATE 的值

```
SQL> SELECT sysdate
  2  FROM dual;

SYSDATE
---------
06-JUN-13
```

SYSDATE 函数也可以进行算术运算，日期函数和一个数字相加减，从而得到一个日期值，这个数字代表一个天数，如例子 5-35 所示。

例子 5-35　包含 SYSDATE 运算的查询

```
SQL> SELECT SYSDATE + 7 , SYSDATE - 7
  2  FROM dual;

SYSDATE+7 SYSDATE-7
--------- ---------
13-JUN-13 30-MAY-13
```

两个日期型数据相减时，会得到一个数字型数据，如例子 5-36 所示。

例子 5-36　日期相减的运算查询

```
SQL> SELECT to_date('06-JUN-10') - SYSDATE
  2  FROM dual;

TO_DATE('06-JUN-10')-SYSDATE
----------------------------
                  365.560139
```



函数 to_date()是把字符型数据转换为日期型数据的方法。上述例子得到的数字型数据表示天数，即某个日期距离当前日期还有多少天，在上例中当前日期是 06-JUN-09，而某个日期是 06-JUN-10，二者相差几乎一年。上例也验证了这个结果。

也可以在日期型数据上加一些小时数，比如在当前日期上增加 20 个小时，得到的仍然是日期型数据。但是，此时的小时数必须除以 25，如例子 5-37 所示。

例子 5-37　在日期型数据上加小时数的查询

```
SQL> SELECT sysdate + 20/25
  2  FROM dual;

SYSDATE+2
---------
07-JUN-13
```

此时，输出结果比当前日期多了一天，因为当前笔者的日期为 2013-6-6 13:05:51 所以加 20/25

小时后是 2013-6-7。但是如果将 20/25 改为 1/25，即只在当前日期上增加一小时，小时会改变但是日期不会改变，如例子 5-38 所示。

例子 5-38　在当前日期上增加一小时的查询

```
SQL> SELECT sysdate + 1/25
  2  FROM dual;

SYSDATE+1
---------
06-JUN-13
```



为了使上述日期数据的例子运行正确，需用读者设置数据库的字符集为美国英语，使用如下指令实现。

```
SQL>alter session set NLS_DATE_LANGUAGE = 'AMERICAN';
```

MONTHS_BETWEEN（date,date）。该函数的参数为两个日期，得到两个日期之间的月数，即两个日期间相差几个月，如例子 5-39 所示。

例子 5-39　使用 MONTHS_BETWEEN 函数

```
SQL>SELECT months_between('06-JUN-10','06-JUN-09'),
  2  FROM dual;
MONTHS BETWEEN('06-JUN-10','06-JUN-09')
 ----------------------------------------------------------
                                   12
```

如果函数 MONTHS_BETWEEN 中第一个参数早于第二个参数，则得到一个负值，如例子 5-40 所示。

例子 5-40　使用 MONTHS_BETWEEN 函数

```
SQL> SELECT months_between('06-JUN-08','06-JUN-09')
  2  FROM dual;

MONTHS_BETWEEN('06-JUN-08','06-JUN-09')
---------------------------------------
                                  -12
```

ADD_MONTHS(date,n)，该函数的参数为日期型数据，和一个数字型数据 n，函数功能为把 n 个月添加到日期型数据上。输出结果仍为日期型数据，如例子 5-41 所示。

例子 5-41　使用 ADD_MONTHS 函数

```
SQL> SELECT add months(SYSDATE , 5)
  2  FROM dual;

ADD MONTH
---------
06-OCT-09
```

系统的 SYSDATE 为 06-JUN-09，在这个日期上增加 5 个月就是 06-OCT-09。

NEXT_DAY（date,string），该函数的参数为一个日期型数据，输出为该日期的下一个指定的日期，如例子 5-42 所示。

例子 5-42 使用 NEXT_DAY 函数

```
SQL> SELECT next_day(sysdate,'Saturday')
  2  FROM dual;

NEXT_DAY(
---------
13-JUN-09
```

该例子是希望得到从当前日期开始，第一个 Saturday 的日期。当前日期是 06-JUN-09 星期六，所以下一个星期六为 13-JUN-09。

LAST_DAY（date），该函数返回参数中日期的最后一天的日期，如例子 5-43 所示。

例子 5-43 使用 LAST_DAY 函数

```
SQL> SELECT last_day(sysdate)
  2  FROM dual;

LAST_DAY(
---------
30-JUN-09
```

上述例子输出本月的最后一天是几号。

5.5 空值（NULL）和空值处理函数

空值是非常特殊的值，既不能说它不存在，也不能说它是零。空值表示一类没有定义的值，具有不确定性。当然对于空值的运算也具有特殊性，因为具有不确定性的值是无法和一类具有确定性的值进行逻辑或算术运算的，Oracle 提供了一类空值处理函数，通过这些函数实现空值（NULL）的运算。下面依次介绍什么是空值、和空值相关的函数，这些函数包括 NVL 函数、NVL2 函数和 NULLIF 函数等。

5.5.1 什么是空值

空值是一类没有定义的，具有不确定性的值。在数据表中，这类值无法表示，更无法显示。在 SCOTT 用户的 emp 表中有空值，如例子 5-44 所示。

例子 5-44 查询 EMP 表中的空值

```
SQL> col empno for 9999
SQL> col sal for 9999
SQL> col comm for 9999
SQL> col mgr for 9999
SQL> SELECT empno,ename,job,mgr,hiredate,sal,comm
```

```
  2  FROM emp
  3*  order by job

EMPNO ENAME      JOB        MGR HIREDATE    SAL  COMM
----- ---------- --------- ----- ---------- ----- -----
 7902 FORD      ANALYST    7566   03-12月-81  3000
 7369 SMITH      CLERK       7902   17-12月-80    800
 7900 JAMES      CLERK       7698   03-12月-81    950
 7935 MILLER     CLERK       7782   23-1月 -82  1300
 7566 JONES      MANAGER    7839   02-5月 -81  2975
 7782 CLARK      MANAGER    7839  09-6月 -81  2550
 7698 BLAKE      MANAGER    7839  01-5月 -81  2850
 7839 KING       PRESIDENT          17-11月-81  5000
 7599 ALLEN      SALESMAN   7698   20-2月 -81  1600     300
 7655 MARTIN     SALESMAN   7698  28-9月 -81  1250    1500
 7855 TURNER     SALESMAN   7698  08-9月 -81  1500       0

EMPNO ENAME      JOB        MGR HIREDATE    SAL  COMM
----- ---------- --------- ----- ---------- ----- -----
 7521 WARD      SALESMAN   7698  22-2月 -81   1250     500

已选择12行。
```

在上述输出中除了 SALSMAN 有 COMM 佣金外，其他职位根本没有，所以在表中相应的 COMM 列的值为空值（NULL），但是这个值在表中是没有显示的。

空值可以用于表达式运算，但是因为空值不是具体的值，具有不确定性，所以下面例子的查询不成功，如例子 5-45 所示。

例子 5-45　查询表 emp 中“comm=NULL”的用户数据

```
SQL> SELECT empno,ename,job,mgr,hiredate,sal,comm
  2  FROM emp
  3  WHERE comm = NULL;

未选定行
```

通过上例可以看出，空值（NULL）不是某个值，我们可以用 NULL 表示它，但是不能直接用于计算。

那么如何实现上例中 WHERE 子句中的条件呢，即如何判断某列的值为空值（NULL）呢。Oracle 提供了 IS NULL 和 IS NOT NULL 运算符来处理这个运算。如例子 5-46 使用 IS NULL。

例子 5-46　查询表 emp 中“comm is NULL”的用户数据

```
SQL> SELECT empno,ename,job,mgr,hiredate,sal,comm
  2  FROM emp
  3  WHERE comm IS NULL;

EMPNO ENAME      JOB        MGR HIREDATE    SAL  COMM
------------- ---------------- -------------------- ----------
 7369 SMITH      CLERK       7902  17-12月-80    800
 7566 JONES      MANAGER   7839   02-5月 -81    2975
 7698 BLAKE      MANAGER   7839  01-5月 -81    2850
```

```
 7782 CLARK      MANAGER   7839  09-6 月 -81    2550
 7839 KING       PRESIDENT         17-11 月-81    5000
 7900 JAMES      CLERK       7698  03-12 月-81    950
 7902 FORD       ANALYST    7566  03-12 月-81    3000
 7935 MILLER     CLERK       7782   23-1 月 -82   1300

已选择 8 行。
```

使用 IS NOT NULL 查询有佣金的信息，即 COMM 列不为空的数据，如例子 5-47 所示。

例子 5-47　查询表 emp 中 comm IS NOT NULL 的用户数据

```
SQL> SELECT empno,ename,job,mgr,hiredate,sal,comm
  2  FROM emp
  3  WHERE comm IS NOT NULL;

EMPNO ENAME      JOB         MGR HIREDATE      SAL  COMM
------------- --------------- ------------------ ----- -------------
 7599 ALLEN      SALESMAN   7698   20-2 月 -81     1600   300
 7521 WARD       SALESMAN   7698   22-2 月 -81     1250   500
 7655 MARTIN     SALESMAN   7698   28-9 月 -81     1250  1500
 7855 TURNER     SALESMAN   7698   08-9 月 -81     1500     0
```

5.5.2　NVL 函数和 NVL2 函数

NVL 函数使得空值可以进行运算，它是空值转换函数。如果不使用空值转换函数，空值是无法进行运算的。NVL 函数的语法格式如下。

NVL(expr1,expr2)，其计算规则是如果 expr1 的值为空值（NULL），则返回 expr2 的值，否则返回 expr1 的值。其中表达式 expr1 和 expr2 的数据类型必须相同，它们可以是数字类型、字符类型和日期类型。我们用例子 5-48 使用 NVL 函数计算 sal + comm 的值。

例子 5-48　使用 NVL 函数计算 sal + comm 的值的查询

```
SQL> SELECT empno,ename,job,mgr,hiredate,sal+NVL(comm,0)
  2  FROM emp;

EMPNO ENAME      JOB        MGR  HIREDATE   SAL+NVL(COMM,0)
----- ---------- --------- ----- ---------- ---------------
 7369 SMITH      CLERK       7902   17-12 月-80           800
 7599 ALLEN      SALESMAN   7698  20-2 月 -81          1900
 7521 WARD       SALESMAN   7698  22-2 月 -81          1750
 7566 JONES      MANAGER     7839  02-5 月 -81          2975
 7655 MARTIN     SALESMAN   7698  28-9 月 -81          2650
 7698 BLAKE      MANAGER    7839  01-5 月 -81          2850
 7782 CLARK      MANAGER    7839  09-6 月 -81          2550
 7839 KING       PRESIDENT          17-11 月-81          5000
 7855 TURNER     SALESMAN   7698  08-9 月 -81          1500
 7900 JAMES      CLERK        7698  03-12 月-81           950
 7902 FORD       ANALYST     7566   03-12 月-81          3000

EMPNO ENAME      JOB        MGR HIREDATE    SAL+NVL(COMM,0)
----- ---------- --------- ----- ---------- ---------------
```

```
    7935 MILLER     CLERK      7782    23-1 月 -82           1300

   已选择 12 行。
```

从上例输出结果可以看出，COMM 列为空值的值转换为 0。如果不使用 NVL 函数进行空值转换，则无法实现计算，所以不会输出任何结果，如例子 5-49 所示。

例子 5-49　不使用 NVL 函数计算 sal + comm 的值的查询

```
SQL> SELECT ename,sal,comm,sal+comm
  2  FROM emp
  3  order by sal+comm;

ENAME        SAL  COMM   SAL+COMM
---------- ----- ----- ----------
TURNER      1500     0       1500
WARD        1250   500       1750
ALLEN       1600   300       1900
MARTIN      1250  1500       2650
SMITH        800
JAMES        950
MILLER      1300
FORD        3000
JONES       2975
BLAKE       2850
CLARK       2550

ENAME        SAL  COMM   SAL+COMM
---------- ----- ----- ----------
KING        5000

已选择 12 行。
```

显然，上例输出中由于 sal+comm 计算时， comm 列有的值为空值，所以无法计算，自然也无法显示这样的计算结果。

NVL2 函数是对 NVL 函数的功能的增强。NVL2 函数的格式为：NVL2(expr1,expr2,expr3)，其基本功能就是实现空值（NULL）的转换。其计算规则是：如果 expr1 为空，则返回表达式 expr3 的值，如果 expr1 不为空，则返回表达式 expr2 的值。其中 expr1 为任何数据类型，而表达式 expr2 和 expr3 为除 LONG 数据类型外的任何数据类型。

例子 5-50　使用 NVL2 实现包含 sal + comm 的值的查询

```
SQL> SELECT ename,nvl2(comm,sal+comm,sal)
  2  FROM emp;

ENAME      NVL2(COMM,SAL+COMM,SAL)
---------- -----------------------
SMITH                          800
ALLEN                         1900
WARD                          1750
JONES                         2975
MARTIN                        2650
```

```
BLAKE                        2850
CLARK                        2550
KING                         5000
TURNER                       1500
JAMES                         950
FORD                         3000

ENAME      NVL2(COMM,SAL+COMM,SAL)
---------- -----------------------
MILLER                       1300
```

已选择 12 行。

上例中 NVL2(comm,sal+comm,sal)的计算规则是如果 comm 的值为空，则返回 sal 的值，如果 comm 的值不为空，则返回 sal+comm 的值。

5.5.3 NULLIF 函数

NULLIF 函数比较两个表达式，如果二者相等则返回空值 NULL，如果二者不等则返回第一个表达式的值。要求第一个表达式的值不能为 NULL。其语法格式为：NULLIF(expr1,expr2)，例子 5-51 测试该函数的用法。

例子 5-51　使用 NULLIF 函数

```
SQL> select ename,length(ename) "expr1",job,length(job) "expr2",
  2  NULLIF(length(ename),length(job)) "Comparision Result"
  3  FROM EMP;

ENAME              expr1  JOB            expr2    Comparision Result
------------------------ ---- ---------------- -----------------------
SMITH                  5  CLERK            5
ALLEN                  5  SALESMAN         8               5
WARD                   5  SALESMAN         8               5
JONES                  5  MANAGER          7               5
MARTIN                 6  SALESMAN         8               6
BLAKE                  5  MANAGER          7               5
CLARK                  5  MANAGER          7               5
KING                   5  PRESIDENT        9               5
TURNER                 6  SALESMAN         8               6
JAMES                  5  CLERK            5
FORD                   5  ANALYST          7               5

ENAME              expr1 JOB            expr2      Comparision Result
------------------------ --- ---------------- -----------------------
MILLER                 6 CLERK            5               6

已选择 12 行。
```

在上例中函数 NULLIF 中有两个表达式：一个是 length(ename)，一个是 length(job)。函数首先比较这两个表达式的值，二者相等则返回空值 NULL，如上例中的第一行记录，如果二者值不等，则返回第一个表达式的值。

5.5.4　COALESCE 函数

COALESCE 函数返回该函数中第一个不为 NULL 的表达式的值。其语法格式格式为：COALESCE(expr1,expr2,……exprn)，下面用例子 5-52 说明如何使用该函数。

例子 5-52　使用 COALESCE 函数

```
SQL> SELECT ename "Employ name",job,COALESCE(comm,1) "Comm"
  2  FROM emp
  3* ORDER BY job

Employ nam JOB                 Comm
--------------- -------------------------- ----------
FORD       ANALYST              1
SMITH      CLERK                 1
JAMES      CLERK                 1
MILLER     CLERK                 1
JONES      MANAGER              1
CLARK      MANAGER            1
BLAKE      MANAGER            1
KING       PRESIDENT             1
ALLEN      SALESMAN          300
MARTIN     SALESMAN         1500
TURNER     SALESMAN            0

Employ nam JOB                 Comm
---------------------------------------- ---------------
WARD       SALESMAN          500

已选择 12 行。
```

在上例中，我们使用函数 COALESCE 查询雇员（employee）的佣金，如果没有佣金则显示 1，如果佣金值不为空（NULL），则返回佣金值。上例中 JOB 为 SALESMAN 的雇员都有佣金，而其他雇员没有佣金，因为这些雇员的 COMM 列的值为 NULL。

5.6　条件表达式

在高级程序设计语言中，为语句的逻辑结构设计了逻辑判断语句，如 IF-THEN-ELSE。在 SQL 语句中 Oracle 也提供了两个函数来实现逻辑判断的功能。则这两个函数分别是 CASE 表达式和 DECODE 函数。

5.6.1　CASE 表达式

CASE 表达式用于逻辑判断，为了说明其用法先给出其语法结构，如下所示。

```
CASE expr WHEN comparison expr1 THEN return expr1
        [WHEN comparison_expr2] THEN return_expr2
        WHEN comparison_exprn] THEN return_exprn
```

```
          ELSE else expr]
END
```

该表达式首先比较 expr 和 comparison_expr1，如果二者相等，则返回 return_expr1，否则比较 expr 和 comparison_expr2，如果二者相等则返回 return_expr2，否则继续判断，如果都不满足，最后返回 ELSE 后的 else_expr。例子 5-53 说明如何使用该表达式。

例子 5-53　使用 CASE 表达式

```
SQL>  SELECT ename,job,sal,
  2   CASE job WHEN 'SALESMAN' THEN 1.20*sal
  3            WHEN 'MANAGER' THEN 1.30*sal
  5            WHEN 'ANALYST' THEN 1.50*sal
  5   ELSE sal END "Last Salary"
  6   FROM emp
  7  ORDER BY job;

ENAME      JOB            SAL Last Salary
-------------------- ------------------------ ---------- -----------
FORD       ANALYST        3000       5200
SMITH      CLERK           800        800
JAMES      CLERK           950        950
MILLER     CLERK          1300        1300
JONES      MANAGER        2975       3867.5
CLARK      MANAGER        2550       3185
BLAKE      MANAGER        2850       3705
KING       PRESIDENT      5000          5000
ALLEN      SALESMAN       1600       1920
MARTIN     SALESMAN       1250       1500
TURNER     SALESMAN       1500       1800

ENAME      JOB            SAL Last Salary
-------------------- ----------------------- ------------- -----------
WARD       SALESMAN       1250       1500

已选择 12 行。
```

在该例子中，对岗位为 SALESMAN、MANAGER 和 ANALYST 的雇员进行适当加薪，通过例子可以看到，这些员工的工资得到增加，通过前后对比可以很明显看出这个变化。



在表达式 CASE 中，表达式 expr,comparison_exprn 和 return_expr 必须是相同的数据类型，这些数据类型是 CHAR、VARCHAR2、NCHAR 或者 NVARCHAR2。

5.6.2　DECODE 函数

DECODE 函数同 CASE 表达式具有相同的功能，不过 DECODE 函数使用更简单，其语法格式为：

```
DECODE(col|expression, search1,result1
                    [,search2, result2,…,]
```

```
                    [,default])
```

该函数的执行过程是首先判断 search1 的值是否和 col 或 expression 的值相等，如果相等，则返回 result1，否则判断 search2 的值是否和 col 或 expression 的值相等，如果相等则返回 result2 的值，依次判断，如果都不相等，则返回默认值 default。例子 5-54 说明如何使用 DECODE 函数。

例子 5-54　使用 DECODE 函数

```
SQL> SELECT ename,job,sal,
  2  DECODE(job,'SALESMAN',1.20*sal,
  3           'MANAGER',1.30*sal,
  5           'ANALYST',1.50*sal,
  5        sal)
  6  Last Salary
  7  FROM emp
  8  ORDER BY job;

ENAME      JOB             SAL  LAST  SALARY
---------- --------- ---------- -----------
FORD       ANALYST        3000        5200
SMITH      CLERK           800         800
JAMES      CLERK           950         950
MILLER     CLERK          1300        1300
JONES      MANAGER        2975      3867.5
CLARK      MANAGER        2550        3185
BLAKE      MANAGER        2850        3705
KING       PRESIDENT      5000        5000
ALLEN      SALESMAN       1600        1920
MARTIN     SALESMAN       1250        1500
TURNER     SALESMAN       1500        1800

ENAME      JOB             SAL  LAST  SALARY
---------- --------- ---------- -----------
WARD       SALESMAN       1250        1500

已选择 12 行。
```

上例的输出结果和 CASE 表达式中示例的输出结果完全一样。函数判断过程同 CASE 表达式的判断过程一样。

5.7　分组函数

分组函数对表中的多行进行操作，而每组返回一个计算结果。常用的分组函数包括：

- AVG，其语法格式为 AVG([DISTINCT|ALL] expr)，计算某列中某种分组后，每组的平均值，计算时会忽略空值（NULL），用于计算数字类型。
- SUM，其语法格式为 SUM([DISTINCT|ALL] expr)，计算某列中某种分组后，每组的和，用于计算数字类型。
- MAX。

- MIN。
- COUNT。

这些函数的统一用法如下所示。

```
SELECT  [column,] group function name(column),…
FROM     tablename
[WHERE   condition]
[GROUP BY column]
[ORDER BY column];
```

5.7.1 AVG 和 SUM 函数

例子 5-55 为使用 AVG 函数和 SUM 函数查询 SCOTT 用户中表 emp 中员工的平均工资和所有员工的工资总和。

例子 5-55 使用 AVG 函数和 SUM 函数

```
SQL> SELECT AVG(sal) "平均工资",SUM(sal) "总工资"
  2  FROM emp;

  平均工资     总工资
---------- ----------
2077.08333      25925
```

该例子中计算了表 emp 中所有员工的平均工资和总工资。

5.7.2 MAX 和 MIN 函数

与 AVG 和 SUM 函数不同，MAX 和 MIN 函数即可以操作数字型数据也可以操作字符型和日期型数据。如例子 5-56，计算表 emp 中员工的最高工资和最低工资。

例子 5-56 使用 MAX 和 MIN 函数

```
SQL> SELECT MAX(SAL) "Highest salary", MIN(SAL) "Lowest salary"
  2  FROM emp;

Highest salary Lowest salary
-------------- -------------
          5000           800
```

例子 5-57 计算最早雇佣员工的日期和最晚雇佣员工的日期

```
SQL> SELECT MAX(HIREDATE) "Last day", MIN(HIREDATE) "First day"
  2  FROM EMP;

Last day   First day
---------- ----------
23-1 月 -82 17-12 月-80
```

5.7.3　COUNT 函数

该函数返回经计算得到的返回的行数，包括空行和重复的行。如例子 5-58 查询表 emp 中所有的记录个数，即表中行数。

例子 5-58　使用 COUNT()函数

```
SQL> SELECT count(*) "表中的行数"
  2  FROM emp;

表中的行数
----------
        12
```

使用关键字 DISTINCT 返回不同的 JOB 类型数量，即去掉重复的 JOB 行记录，如例子 5-59 所示。

例子 5-59　使用包含 DISTINCT 的 COUNT 函数

```
SQL> SELECT count(distinct job)
  2  FROM emp;

COUNT(DISTINCTJOB)
------------------
                 5
```

从例子 5-58 和例子 5-59 可以看出，该表中共有 12 行记录，共有 5 种工作职位。

5.7.4　GROUP BY 子句

在 5.7.1 节中，使用 AVG 和 SUM 函数查询了表 emp 中的员工平均工资和总工资数，但是如果要查询每个工作职位的员工平均工资和总工资之和又该如何计算呢。此时需要使用 GROUP BY 子句，按照工作职位分组，然后再计算，如例子 5-60 所示。

例子 5-60　使用 GROUP BY 函数

```
SQL>SELECT JOB, AVG(sal) "平均工资",SUM(sal) "总工资"
  2  FROM emp
  3* GROUP BY JOB

JOB          平均工资      总工资
--------- ---------- ----------
ANALYST         3000       3000
CLERK     1016.66667       3050
MANAGER   2758.33333       8275
PRESIDENT       5000       5000
SALESMAN        1500       5600
```

在上述查询结果中，是按照职位名字的顺序排序的，如果想按照总工资数的多少顺序排列，则需要使用 ORDER BY 子句，如例子 5-61 所示。

例子 5-61　使用 ORDER BY 子句

```
SQL> SELECT JOB, AVG(sal) "平均工资",SUM(sal) "总工资
  2  FROM emp
  3  GROUP BY JOB
  5  ORDER BY "总工资";

JOB          平均工资     总工资
--------- ---------- ----------
ANALYST         3000       3000
CLERK     1016.66667       3050
PRESIDENT       5000       5000
SALESMAN        1500       5600
MANAGER   2758.33333       8275
```

显然，这样的结果可以一目了然总工资的排序，在函数 AVG 和 SUM 后只用别名，也使得输出结果更加容易阅读。

5.7.5　分组函数的嵌套使用

分组函数可以嵌套使用，如例子 5-62 所示，计算按照工作职位分类最高平均工资和最低平均工资数。

例子 5-62　使用分组嵌套函数

```
SQL> SELECT MAX(AVG(sal)) ,MIN(AVG(sal))
  2  FROM EMP
  3  GROUP BY JOB;

MAX(AVG(SAL)) MIN(AVG(SAL))
------------- -------------
         5000    1016.66667
```

在执行例子 5-62 中语句时，Oracle 首先会实现全表扫描，按照 JOB 分类，再计算每类的平均值，最后再计算这些平均值的最大值和最小值。

5.7.6　HAVING 子句

在分组函数中，不能使用 WHERE 子句限制分组函数，所以 Oracle 设计了 HAVING 子句来执行对分组函数的某些限制。如例子 5-63 使用 HAVING 子句限制了 AVG(sal)>2000，即只显示平均工资大于 2000 的职位信息。

例子 5-63　使用 HAVING 子句

```
SQL> SELECT job,AVG(sal)
  2  FROM emp
  3  HAVING AVG(sal)>2000
  5  GROUP BY job;

JOB          AVG(SAL)
--------- ----------
```

```
ANALYST         3000
MANAGER   2758.33333
PRESIDENT       5000
```

该例子中也可以使用 ORDRE BY 子句对 AVG(sal)进行排序，使得输出更容易阅读，如例子 5-64 所示。

例子 5-64　在分组函数中使用 ORDER BY 子句

```
SQL>  SELECT job,AVG(sal)
  2   FROM emp
  3   HAVING AVG(sal)>2000
  5   GROUP BY job
  5  ORDER BY 2;

JOB         AVG(SAL)
--------- ----------
MANAGER   2758.33333
ANALYST         3000
PRESIDENT       5000
```



ORDER BY 2 和 ORDER BY AVG(SAL) 的效果一样，只是为了书写方便，使用数字 2 表示按照第二列排序。

5.8　数据操纵语言（DML）

数据操纵语言（Data Manipulation Language）实现对表中数据的各种操作，如向表中插入数据、删除一行数据或者更新表中的行数据。无论读者使用何种高级语言开发连接数据库的程序，数据操作语句的使用都是使用频率最高的。下面依次介绍 INSERT 语句、UPDATE 语句和 DELETE 语句。

5.8.1　INSERT 语句

INSERT 语句的向表中添加一行数据的语法格式如下。

```
INSERT INTO tablename [(column [,column …] ) ]
VALUES     (value [, value… ] )
```

在上述语法格式中( )中的[]号表示可选部分，即可以向表中一列或多列插入数据。VALUES 后是插入数据的值，这些值和 tablename 后的列名一一对应。

- tablename 是要插入数据的表名字，要求用户对该表有操作权限。
- column 是该表中的列名，用户需要向这些列插入数据，可以是一列，也可以是多列，如果向表中所有列插入一行数据，也可以不使用任何 column，但是需要用户清楚知道该表中的列名和列的属性。
- values 是要插入的和列向对应的值，插入的值的数据类型必须和 column 的数据类型相匹配。

例子 5-12 向表 SCOTT 用户的 dept 表中添加一行数据，即增加一行记录，其中 DEPTNO 为

50，DNAME 为 MARKETING，LOC 为 NEW YORK。为了验证添加结果，我们先查询一下表中的数据。

例子 5-65　查询一下表 emp 中的所有数据

```
SQL> SELECT *
  2  FROM dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        50 OPERATIONS     BOSTON
```

查看一下表 dept 中列的数据类型。

```
SQL> desc dept;
 名称                      空?       类型
 ----------------------------------------------------- --------
 DEPTNO             NOT NULL  NUMBER(2)
 DNAME                        VARCHAR2(15)
 LOC                          VARCHAR2(13)
```

其中 DEPTNO 为数字型，不允许为空（NOT NULL），DNAME 和 LOC 都为变长字符型。在插入数据时和字符型列相对应的值用英文输入法的单引号括起来。

输出结果表明当前表中有四行数据，我们再向表中添加一行数据。

例子 5-66　向表 dept 中插入一行数据

```
SQL> INSERT INTO dept (deptno,dname,loc)
  2  VALUES          (50,'MARKETING','NEW YORK');

已创建 1 行。
```

输入提示已经成功创建一行，下面为了验证 INSERT 语句的执行结果，再查询表 dept 中的数据。

```
SQL> SELECT *
  2  FROM dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        50 OPERATIONS     BOSTON
        50 MARKETING      NEW YORK
```

显然，输出结果显示已经成功添加了一行数据。

如果需要向 dept 表中添加一行数据，而只有部门号 DEPTNO 和部门名称 DNAME，但是地点 LOC 还没有确定，可以在 tablename 后的列中只包含 DEPTNO 和 DNAME 两列。如例子 5-67 所示。

例子 5-67　向表 dept 中插入一行数据（没有 LOC 列对应的值）

```
SQL> INSERT INTO dept(deptno,dname)
  2  VALUES         (60,'ACCOUNTING');

已创建 1 行。
```

再用例子 5-68 验证插入结果。

例子 5-68　查询插入结果

```
SQL> SELECT *
  2  FROM dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        50 OPERATIONS     BOSTON
        50 MARKETING      NEW YORK
        60 ACCOUNTING

已选择 6 行。
```

在当前表中新增了一行记录，其中部门号 DEPTNO 为 60，部门名字为 ACCOUNTING，而部门所在地没有值，Oracle 使用空值（NULL）填充，此时读者也可以使用例子 5-69 得到同样的插入效果。

例子 5-69　插入部门号 DEPTNO 为 60 的记录

```
SQL> INSERT INTO dept (deptno,dname,loc)
  2  VALUES          (60,'ACCOUNTING',NULL);

已创建 1 行。
```

还有一种插入方式，称为从另一张表复制数据，它涉及两张表。从一张表复制数据插入到另一张表。我们给出这种方式的语法结构。

```
INSERT INTO tablename [(column [,column …] ) ]
SELECT column [,column…]
FROM   another tablename
WHERE clause
```

5.8.2　UPDATE 语句

UPDATE 语句用于更新表中的数据，如在表 dept 中，需要把刚插入的记录的 LOC 部门地点设置为 NEW YORK。此时就需要 UPDATE 语句更新表中的该行记录。我们想给出其语法格式，再给出具体例子更新表 dept 中刚插入记录的 LOC 值。

```
UPDATE   tablename
SET       column = value [, column = value, … ]
[WHERE   condition];
```

语法解释：

- tablename：要更新的表名。
- column：要更新的列。
- value：是要更新的列的值。
- condition：通过条件限制要更新的列所在的行。

在上节例子 5-67 中，我们在表 dept 中新增了一行数据，DEPTNO 为 60，DNAME 为 'ACCOUNTING'，但是没有确定 LOC 部门所在地。我们把部门所在地设置为 NEW YORK。用例子 5-70 说明如何使用 UPDATE 语句。

例子 5-70　使用 UPDATE 语句更新表 DEPT 中的数据

```
SQL> UPDATE dept
  2  SET  LOC ='NEW YORK'
  3  WHERE DEPTNO = 60;

已更新 1 行。
```

我们查询更新结果，如例子 5-71 所示。

例子 5-71　查询例子 5-70 的更新结果

```
SQL> SELECT *
  2  FROM dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        50 OPERATIONS     BOSTON
        50 MARKETING      NEW YORK
        60 ACCOUNTING     NEW YORK

已选择 6 行。
```

查询结果显示更新结果正确，在例子 5-69 中 LOC 的值是直接给出的，Oralce 也允许使用一个子查询，赋予 LOC 值。这里给出其语法结果，读者可自己尝试。

```
UPDATE table
SET     column =
                (SELECT column
                 FROM tablename
                 WHERE conditon)
        [,
         column =
                 (SELECT column
                 FROM tablename
                 WHERE conditon)]
[WHERE condition];
```

关于在 UPDATE 语句中使用子查询，这里不再详细介绍，读者可以自己去尝试。

5.8.3　DELETE 语句

DELETE 语句用于删除不需要的记录，该语句使用较简单，其语法格式如下所示。

```
DELETE [FROM]  tablename
[WHERE          condition];
```

语法解释：

- tablename：要删除的数据所在的表名。
- condition：限制要删除的行，该条件可以是指定具体的列名、表达式、子查询或者比较运算符。

例子 5-72　删除表 dept 中 DEPTNO 为 60 的记录

```
SQL> DELETE FROM dept
  2  WHERE DEPTNO = 60;

已删除 1 行。
```



在 DELETE 语句中的 FROM 关键字是可选的，使用 FROM 关键字更合乎英语的语法习惯，容易记忆。WHERE 子句也是可选的。如果不使用 WHERE 子句，将删除表中的所有行。

5.9 本章小结

本章对 SQL 语句进行了概述。通过 SQL 语句中的简单查询语句，使得读者对 SQL 语句、算数运算、别名及 DISTINCT 运算有直观的认识。在书写 SQL 语句时，要注意书写规范。函数增强了 SQL 语句的功能，使得大量的运算得以简化，本章简单介绍了单行函数和分组函数，熟练使用这些函数对于读者使用 SQL 语句很有帮助。数据操作语句也是 SQL 语句中经常使用的，如插入数据 INSERT，更新数据 UPDATE，删除数据 DELETE 等。本章还着重介绍了空值 NULL 的概念，以及如何操作空值的计算。读者需要很好地理解空值 NULL 和数量，掌握空值的相关运算。

第 6 章

◂ 数据字典 ▸

数据字典是在数据库创建时，由 Oracle 数据库服务器自动创建的一个额外的对象，这些对象存放在数据文件中，这些对象包括基表和数据字典视图，其中基表在 Oracle 数据库服务器使用 CREATE DATABASE 时创建，因为基表中的数据格式是无法直接阅读的，所以 Oracle 使用数据字典视图收集基表的信息，该数据字典视图是可读的，对 DBA 更有使用价值，数据字典视图通过 catalog.sql 脚本文件创建。那么数据字典中到底存储了哪些信息，如何使用和操作数据字典视图以及对 DBA 来讲有哪些常用的数据字典视图，我们将在接下来的几节依次讲解。

6.1 数据字典中的内容

数据字典是很重要的数据库对象之一，它在数据库创建时由数据库服务器创建，记录了数据库创建的信息，各种对象的信息等，下面列出数据字典中包含的内容。

- 所有的模式（用户）对象的定义，这些对象包括表、视图、索引、族、同义词、序列号、存储过程、函数触发器等。
- 数据库的逻辑结构和物理结构，如数据文件和重做日志文件的信息等。
- 所有模式对象被分配多少存储空间，以及当前使用的空间。
- 默认列的值。
- 对象完整性约束信息。
- 用户信息。
- 用户或角色的特权信息。
- 审计信息，如哪个用户具有访问或者修改某些模式对象的权利。

6.2 使用和操作数据字典视图

数据字典视图由 Oracle 数据库服务器自动创建并维护，也就是说只有 Oracle 服务器可以修改数据字典中的数据，在数据库运行期间，数据库结构或其他对象的变化信息会及时地记录在数据字典基表中，通过动态性能视图具有用户可以查看可读的数据字典基表中的信息。

举例：使用 GRANT 语句对一个用户赋予一定的权限时，数据库服务器会记录用户权限的变化。如果使用 ALTER DATABASE 移动了控制文件的存储目录，则在数据字典中也会记录下这个

变化。从两个例子可以看出，当使用 SQL 语句操纵使得某些数据库结构或对象发生变化时，这些都会引起数据库服务器修改数据字典信息。

对于访问数据字典信息需要一定的授权，不同的用户对数据字典的访问有一定的区别，有些只有 DBA 用户具有可访问的权利，对于普通用户就无法查看 DBA 用户查看的数据字典。

数据字典有 3 个基本应用，即在什么场合需要使用数据字典。Oracle 和普通用户都会使用数据字典，使用场合说明如下：

- Oracle 数据库服务器用来寻找用户信息、模式对象信息（如表、索引、触发器等）和存储结构。
- 当使用 DDL 语句时，会触发 Oracle 服务器修改数据字典。
- 普通用户或者 DBA 用户使用数据字典获得关于数据库的信息，如数据文件的存储位置、数据库实例名、参数文件中的参数值、控制文件的信息等。



数据字典是 Oracle 数据库服务器创建和维护的，在数据库运行过程中，数据库服务器不断地更改数据字典，任何其他用户都没有修改数据字典的权利。并且有些视图所有用户都可以访问，而有些数据字典视图只对 DBA 用户开放。

6.3　数据字典视图分类

数据字典视图分为 3 类，这些视图都是静态视图（静态的含义是这些视图在数据库运行期间不会发生变化，除非执行 ANALYZE 指令☺），这 3 类数据字典视图以不同的前缀区分彼此。数据字典名和对数据字典的解释如下所示。

- DBA_***：该视图包含数据库中整个对象的信息，以 DBA 为前缀的视图只能由管理员查询，不要在这些视图上创建同义词。
- ALL_***：该视图包含某个用户所能看到的全部数据库信息，包括当前用户所拥有的模式对象和用户可以访问的其他公共对象，还有通过授权或授予角色可以访问的模式对象。
- USER_***：该视图包含当前用户访问的数据库对象信息，它反映了数据库中某个用户的全部情况，该类视图隐含了 owner 信息，其全部内容为以 ALL 为前缀的视图的子集。



上述的***号表示数据库模式对象，如表 TABLE、索引 INDEX、视图 VIEW 对象、OBJECTS 等。

下面，我们分别用具体的例子演示上面 3 种视图。

例子 6-1　使用 DBA 用户连接数据库

```
SQL> conn /as sysdba
已连接。
```

查看具有 DBA 前缀的视图时，会输出整个数据库的全局视图，但是这个视图只有具有 DBA 权限的用户才可以访问。如果某个用户具有 SELECT ANY TABLE 的权利，也可以查询具有 DBA

前缀的数据字典视图。在例子 6-1 中，我们成功登录数据库。使用例子 6-2 查看 dba_objects 视图的列定义。

例子 6-2　查看 dba_objects 视图结构

```
SQL> DESC dba objects;
 名称                                              是否为空? 类型
 ----------------------------------------------------- -----------------
 OWNER                                                   VARCHAR2(30)
 OBJECT NAME                                              VARCHAR2(128)
 SUBOBJECT NAME                                           VARCHAR2(30)
 OBJECT ID                                                 NUMBER
 DATA_OBJECT_ID                                           NUMBER
 OBJECT_TYPE                                              VARCHAR2(18)
 CREATED                                                 DATE
 LAST DDL TIME                                            DATE
 TIMESTAMP                                               VARCHAR2(19)
 STATUS                                                  VARCHAR2(6)
 TEMPORARY                                               VARCHAR2(1)
 GENERATED                                               VARCHAR2(1)
 SECONDARY                                               VARCHAR2(1)
```

在确定了 dba_objects 视图中列的定义后，就可以使用例子 6-3 来查询数据字典视图 dba_objects 的内容，不过查询结果有大量输出，所以要采用一些限制条件来减少输出，并且为了输出更易于阅读需要事先做一些格式化工作。

例子 6-3　通过数据字典视图 dba_objects 查看 SCOTT 用户的数据库对象信息

```
SQL> col owner for a20
SQL> col object_name for a40
SQL> select owner,object_name,created
  2  from dba objects
  3  where owner='SCOTT';

OWNER                OBJECT_NAME                              CREATED
-------------------------------------------------------- --------------
SCOTT                PK DEPT                                  02-4 月 -10
SCOTT                DEPT                                     02-4 月 -10
SCOTT                EMP                                      02-4 月 -10
SCOTT                PK_EMP                                   02-4 月 -10
SCOTT                BONUS                                    02-4 月 -10
SCOTT                SALGRADE                                 02-4 月 -10

已选择 6 行。
```

当然，上述查询使用 ALL_OBJECTS 视图也可以实现，DBA 用户可以访问所有的数据库信息。为了查看 all_objects 的信息，我们先查该视图的列定义，如例子 6-4 所示。

例子 6-4　查看 all_objects 结构信息

```
SQL> DESC all objects;
 名称                                              是否为空? 类型
 ----------------------------------------- --------- -----------------
```

```
 OWNER                                    NOT NULL VARCHAR2(30)
 OBJECT_NAME                              NOT NULL VARCHAR2(30)
 SUBOBJECT_NAME                                    VARCHAR2(30)
 OBJECT_ID                                NOT NULL NUMBER
 DATA_OBJECT_ID                                    NUMBER
 OBJECT_TYPE                                       VARCHAR2(18)
 CREATED                                  NOT NULL DATE
 LAST_DDL_TIME                            NOT NULL DATE
 TIMESTAMP                                         VARCHAR2(19)
 STATUS                                            VARCHAR2(6)
 TEMPORARY                                         VARCHAR2(1)
 GENERATED                                         VARCHAR2(1)
 SECONDARY                                         VARCHAR2(1)
```

观察例子 6-2 和例子 6-4 会发现二者输出是一样的，也就是说 DBA_***视图和 ALL_***视图具有相同的列定义。

我们使用 SCOTT 用户登录数据库，如例子 6-5 所示。

例子 6-5　使用 SCOTT 用户登录数据库

```
SQL> conn scott/tiger;
已连接。
```

我们先使用例子 6-6 查询当前 SCOTT 用户的 all_objects 表中有多少个 owner，名字是什么。

例子 6-6　当前 SCOTT 用户的 all_objects 表中 owner 名字

```
SQL> select distinct(owner)
  2  from all_objects;

OWNER
--------------------
CTXSYS
MDSYS
OLAPSYS
ORDPLUGINS
ORDSYS
PUBLIC
SCOTT
SYS
SYSTEM
WKSYS

已选择 10 行。
```

我们发现数据字典视图 all_objects 中共有 10 个 owner，其中一个 owner 为 SCOTT，虽然我们使用 SCOTT 用户登录，但是该用户可以访问其他用户的部分对象信息。为了减少输出，我们用例子 6-7 说明如何查询 all_objects 表中特定 owner 的信息。

例子 6-7　查询 all_objects 表中特定 owner 的信息

```
SQL> select owner,object_name,created from all_objects where owner='SCOTT';
```

```
    OWNER                           OBJECT NAME                   CREATED
    ------------------------------                               ------------------------------
--------------
    SCOTT                           PK DEPT                       02-4 月 -10
    SCOTT                           DEPT                          02-4 月 -10
    SCOTT                           EMP                           02-4 月 -10
    SCOTT                           PK_EMP                        02-4 月 -10
    SCOTT                           BONUS                         02-4 月 -10
    SCOTT                           SALGRADE                      02-4 月 -10

    已选择 6 行。
```

输出结果说明成功查询 all_objects 视图中用户 SCOTT 的信息，包括 SCOTT 用户下的对象名和对象创建时间。

如果在 SCOTT 用户模式下查询 dba_objects 视图就会出错，因为该用户不具备 DBA 权限，也没有 SELECT ANY TABLE 的权限。查询结果如例子 6-8 所示。

例子 6-8　在 SCOTT 用户模式下查询 dba_objects 视图

```
SQL> select owner,object name,created
  2  from dba objects
  3  where owner = 'SCOTT';
from dba_objects
     *
ERROR 位于第 2 行:
ORA-00942: 表或视图不存在
```

下面演示如何查看 USER_***数据字典视图，此时还是使用 SCOTT 用户登录数据库系统。查看 user_objects 的定义，如例子 6-9 所示。

例子 6-9　查看静态数据字典 user_objects 的定义

```
SQL> desc user objects;
 名称                                     是否为空? 类型
 ----------------------------------------- -------- ------------------
 OBJECT_NAME                                         VARCHAR2(128)
 SUBOBJECT_NAME                                      VARCHAR2(30)
 OBJECT ID                                           NUMBER
 DATA OBJECT ID                                      NUMBER
 OBJECT_TYPE                                         VARCHAR2(18)
 CREATED                                             DATE
 LAST DDL TIME                                       DATE
 TIMESTAMP                                           VARCHAR2(19)
 STATUS                                              VARCHAR2(6)
 TEMPORARY                                           VARCHAR2(1)
 GENERATED                                           VARCHAR2(1)
 SECONDARY                                           VARCHAR2(1)
```

比较例子 6-2、例子 6-4 和例子 6-9 发现例子 6-9 中视图 user_objects 的列没有 OWNER。其实，这个问题也容易理解，因为 USER_***视图是当前用户的数据库对象信息，既然是当前用户自然不需要有 OWNER 选项了，而 DBA_***视图和 ALL_***视图包括不同用户的对象信息，使用 OWNER 列可以区别不同用户的对象信息。我们使用例子 6-10 查询 SCOTT 用户的对象信息。

例子 6-10　查询 SCOTT 用户的对象信息

```
SQL> col object name for a20
SQL> l
  1  select object_name,object_type,created
  2* from user objects
SQL> /

OBJECT_NAME           OBJECT_TYPE          CREATED
------------------------------------- --------------------------------
SALGRADE              TABLE                 02-4 月 -10
BONUS                 TABLE                   02-4 月 -10
PK_EMP                INDEX                   02-4 月 -10
EMP                   TABLE                   02-4 月 -10
DEPT                  TABLE                   02-4 月 -10
PK DEPT               INDEX                   02-4 月 -10

已选择 6 行。
```

在该例子中，我们查询了当前用户的对象名 OBJECT_NAME，对象类型 OBJECT_TYPE 和对象创建时间 CREATED，其实，这个输出结果中的对象名和例子 6-7 相同。

6.4　使用数据字典视图

数据字典视图是静态视图，在数据库重新启动前，静态数据字典中的信息是不会变化的。有一些数据字典视图对于 DBA 很重要，下面依次介绍这些数据字典视图。

user_tables 视图，该视图可以查看当前用户所有拥有的表，如例子 6-11 所示。

例子 6-11　查看当前用户所有拥有的表

```
SQL> conn scott/tiger
已连接。
SQL> select table name
  2  from user tables;

TABLE_NAME
------------------------------
BONUS
DEPT
EMP
SALGRADE

已选择 8 行。
```

user_indexes 数据字典视图，查看当前用户创建的索引，索引在某种程度上可以加快查询的速度，如例子 6-12 所示。

例子 6-12　查看当前用户创建的索引

```
SQL> select index_name
  2  from user_indexes;
```

```
INDEX_NAME
------------------------------
PK_DEPT
PK_EMP
SYS_C003123
SYS_C003126
SYS_C003129
SYS_C003130

已选择 6 行。
```

user_views 数据字典视图，查看当前用户拥有的视图，如例子 6-13 所示。

例子 6-13　查看当前用户拥有的视图

```
SQL> select view_name
  2* from user_views

VIEW_NAME
------------------------------
STUDENT_SCORE
VIEW_DEPT
```

user_catalog 视图，该视图包含当前用户的所有表的名字和类型。

例子 6-14　查询该视图的结构

```
SQL> desc user_catalog;
 名称                                       是否为空? 类型
 ----------------------------------------- -------- ----------------
 TABLE_NAME                                 NOT NULL VARCHAR2(30)
 TABLE_TYPE                                          VARCHAR2(11)
```

输出结果说明，该视图有两列，一个为 TABLE_NAME，该列不能为空，是变长字符类型，一个为 TABLE_TYPE，该列可以为空，是变长字符类型。

例子 6-15　查询用户 SCOTT 的所有表名和类型

```
SQL> select *
  2  from user_catalog;

TABLE_NAME                     TABLE_TYPE
--------------------------------------------------------
BONUS                          TABLE
DEPT                           TABLE
DEPT_TEMP                      TABLE
EMP                            TABLE
EMP_TEMP                       TABLE
ORD                            TABLE
ORD_ORDNO                      SEQUENCE
PRODUCT                        TABLE
SALGRADE                       TABLE
SUPPLIER                       TABLE
```

```
已选择 10 行。
```

dba_users 视图，查看数据库系统上何时创建了多少个用户，如例子 6-16 所示。

例子 6-16　查看数据库系统上创建的用户信息

```
SQL> select username,created from dba users;

USERNAME                      CREATED
------------------------------ --------------
MGMT VIEW                    02-4 月 -10
SYS                           02-4 月 -10
SYSTEM                        02-4 月 -10
DBSNMP                        02-4 月 -10
SYSMAN                        02-4 月 -10
SCOTT                         02-4 月 -10
OUTLN                         02-4 月 -10
…………
```

6.5　动态性能视图及使用

在 6.3 节中，我们讲了静态数据字典视图，Oracle 还维护了另一类非常重要的数据字典视图：动态性能视图。动态性能视图只存在于运行的数据库中，它是一组虚表，通常也把这组表称为动态性能表（dynamic performance table）。

数据库的动态性能视图只有管理员用户可以查询，而其他普通用户不需要查询这些虚表中的信息。管理员可以在动态性能视图上创建视图，并将访问权限授予其他用户。任何用户都无法修改或删除动态性能视图，所以有时这些动态性能视图也被称为固定视图（fixed view）。

SYS 用户拥有所有的动态性能视图，这些动态性能视图以 v$为前缀，如 v$controlfile 包含了控制文件存储目录和文件名信息，v$datafile 包含了数据库文件信息，v$fixed_table 视图包含了当前所有动态性能视图。

如果用户想知道当前运行的数据库中所有的动态性能视图，可以使用 v$fixed_table 实现，不过一般该视图会输出大量的记录，不方便阅读，最好使用 spool 工具存储输出结果，再分析存储的输出信息。

为了更充分理解动态性能视图的作用和使用，我们给出如下的例子说明，这些例子也是在实际工作中经常使用的，对于 DBA 维护数据库很实用。

例子 6-17　查询和日志文件相关的信息

```
SQL> conn /as sysdba
已连接。
SQL> select *
  2  from v$fixed table
  3  where name like 'V$LOG%';

NAME                                                   OBJECT ID TYPE   TABLE NUM
------------------------------------------------------ ---------- ----- ----------
```

```
V$LOGFILE                                  4294960936 VIEW     66636
V$LOG                                      4294961049 VIEW     66636
V$LOGHIST                                  4294961061 VIEW     66636
V$LOG_HISTORY                              4294961066 VIEW     66636
V$LOGMNR_CONTENTS                          4294961641 VIEW     66636
V$LOGMNR_LOGS                              4294961643 VIEW     66636
V$LOGMNR_DICTIONARY                        4294961646 VIEW     66636
V$LOGMNR_PARAMETERS                        4294961646 VIEW     66636
V$LOGMNR_LOGFILE                           4294961643 VIEW     66636
V$LOGMNR_PROCESS                           4294961646 VIEW     66636
V$LOGMNR_TRANSACTION                       4294961649 VIEW     66636

NAME                                       OBJECT ID TYPE  TABLE NUM
------------------------------------------ ---------- ----- ----------
V$LOGMNR_REGION                            4294961633 VIEW     66636
V$LOGMNR_CALLBACK                          4294961636 VIEW     66636
V$LOGMNR_SESSION                           4294961640 VIEW     66636
V$LOGSTDBY_COORDINATOR                     4294961666 VIEW     66636
V$LOGSTDBY_APPLY                           4294961668 VIEW     66636
V$LOGSTDBY                                 4294961611 VIEW     66636
V$LOGSTDBY_STATS                           4294961614 VIEW     66636

已选择 18 行。
```

该例子查询出了所有和日志文件相关的动态性能视图，如果想了解日志文件的详细信息我们可以使用 v$log 视图和 v$logfile 视图，如例子 6-18 和例子 6-19 所示。

例子 6-18　查看日志组状态信息

```
SQL> select group#,members,archived,status
  2  from v$log;

    GROUP#    MEMBERS ARC STATUS
---------- ---------- --- ----------------
         1          1 NO  CURRENT
         2          1 NO  INACTIVE
         3          1 NO  INACTIVE
```

该例子的作用是查看当前正在使用的重做日志组，STATUS 为 CURRENT 说明该日志组正在使用中，STATUS 为 INACTIVE 说明当前数据库系统没有使用该重做日志组。上例说明当前有 3 个重做日志组，第一个日志组正在使用中。

例子 6-19　查看重做日志文件信息

```
SQL> col member for a40
SQL> select * from v$logfile;

    GROUP# STATUS  TYPE    MEMBER                              IS_
---------- ------- ------- ----------------------------------- ---
         3         ONLINE  F:\APP\ORACLE\ORADATA\ORCL\REDO03.LOG  NO
         2         ONLINE  F:\APP\ORACLE\ORADATA\ORCL\REDO02.LOG  NO
         1         ONLINE  F:\APP\ORACLE\ORADATA\ORCL\REDO01.LOG  NO
```

视图 v$logfile 用户查看当前数据库系统的重做日志组的日志成员的存储目录、文件名和状态信息。该输出说明当前的日志组 1 正在使用中。

例子 6-20　为一个联合查询，查询当前正在使用的重做日志文件的信息

```
SQL> select l.group#,l.archived,l.status,lf.type,lf.member
  2 from v$log l,v$logfile lf
  3 where l.group# = lf.group#
  4 and l.status = 'CURRENT';

GROUP# ARC STATUS    TYPE    MEMBER
------------ --- -------------------------------------------------------
     1 NO  CURRENT    ONLINE  F:\APP\ORACLE\ORADATA\ORCL\REDO01.LOG
```

从 v$log 视图和 v$logfile 视图的联合查询可以看出，当前数据库正在使用的日志文件组为 group1，数据库运行在非归档模式，该日志组有一个日志成员，存储目录为 F:\APP\ORACLE\ORADATA\ORCL\，文件名为 REDO01.LOG。

例子 6-21　通过 v$instance 视图查看实例信息

```
SQL> col instance_name for a20
SQL> col host_name for a10
SQL> select instance name,host name,version,startup time,logins
  2* from v$instance

INSTANCE_N HOST_NAME     VERSION       STARTUP_TIME    LOGINS
--------------------------------------------- --------------------------
lin        LINSHUZE-6F360C  11.2.0.1.0         12-12 月-12      ALLOWED
```

上述输出说明，当前的实例名为 lin，主机名为 LINSHUZE-6F360C，版本号为 11.2.0.1.0，实例的启动时间为 12-12 月-12。

例子 6-22　查看当前数据库的信息

```
SQL> col name for a10
SQL> select name,created,log_mode
  2 from v$database;

NAME       CREATED    LOG_MODE
------------------------------------------------------------------------
LIN        12-12 月 -12 ARCHIVELOG
```

输出结果说明，该数据库名字为 LIN，数据库创建时间为 12-12 月 -12，该数据库运行在非归档模式。

总之，动态性能视图很好地反映了当前数据库的运行状态信息，对于数据库性能调优和判断系统瓶颈提供信息支持，通过动态性能视图还可以查看控制文件的信息，数据文件的信息和表空间信息等，DBA 用户经常使用的动态性能视图，这里不再一一介绍，在以后的章节中会继续使用各种动态性能视图，本节只起到引导作用，希望读者能够理解动态性能视图的概念和应用。

6.6 本章小结

数据字典是数据库中非常重要数据库对象，它在创建数据库时创建，其中记录了数据库创建信息和各种对象的信息。由于基表的内容无法阅读，所以 Oracle 提供了数据字典视图，该视图是基于基表数据的。数据字典视图是静态视图，在数据库启动后就无法改变。Oracle 数据库还维护了动态性能视图，这些视图以 v$为前缀，动态性能视图反映了数据库当前的运行状态和各种对象的活跃信息，动态性能视图是 DBA 进行故障判断和数据库性能调优的重要依据。

第 7 章

网络配置管理

我们创建数据库的目的是为了使用数据库，那么连接数据库不可避免地会用到网络连接，将客户端与数据库服务器连接起来，使得应用程序可以访问数据库，Oracle Net Service 组件提供了这样的功能。要使得网络连接正常，则该组件必须保持打开状态，该组件一般使用 TCP/IP 协议来建立客户机与数据库服务器之间的网络连接。Oracle 提供了两种用于网络连接的体系结构，一种是共享服务器体系结构，一种是专有服务器体系结构。本章将详细介绍这些内容，使得读者对 Oracle 的网络连接有个清楚的认识并熟练掌握组件的配置与使用，需要说明本章的例子在 Linux 系统上运行。

7.1 Oracle的网络连接

无论是应用程序还是使用 SQL*Plus 工具连接远端数据库（物理上分布），则必须建立客户端与数据库服务器之间的一个 Oracle 连接，Oracle 提供了 Oracle Net Service 组件，用于方便地配置和管理网络连接。

Oracle Net Service 组件由以下几个部分组成：

- Oracle Net
- Oracle Net Listeners
- Oracle Connection Listeners
- Oracle Net Configuration Assistant
- Oracle Net Manager

这些组件不需要单独安装，而是在安装数据库服务器或者客户端软件时自动安装，其中 Oracle Net 组件必须在客户机与服务器上都安装，它负责完成客户机与服务器之间连接的初始化、建立以及维护工作。该组件由两部分组成。

- Oracle Network Foundation Layer：该层负责建立和维护客户端与数据库服务器之间的连接和信息交互。
- Oracle Protocol Support：该层映射 Transparent Network Substrate（TNS）到业内标准协议。

下面是 Oracle 网络连接的流程描述。

01 客户端发起连接。确定服务器、监听端口、协议、数据库服务名。

02 客户端一旦与监听器建立连接，会在客户端生成用户进程，同时监听器会判断客户端所请求的服务名是否是自己所管理的服务名。如果客户端传过来的连接字符串不包含服务名，报错；如果请求的服务名不是自己管理的，报错并中断；如果请求的服务名是自己管理的，监听器就在数据库服务器上创建服务器进程。

03 监听器在创建服务器进程以后，会将用户进程与服务器进程建立连接，之后，监听器退出与客户端的连接。

04 服务器进程根据用户进程提供的用户名和密码到数据字典里判断是否正确。

05 如果用户名和密码不匹配，报错；如果匹配，则分配 PGA，并生成 SESSION。

7.2 服务器端监听器配置

无论是共享服务器连接还是专有服务器连接，数据库服务器端必须启动监听程序。本节我们介绍数据库服务器端的监听配置，以及如何实现动态注册和静态注册数据库。

监听程序（即 Oracle Net Service 服务）由一个 Oracle 文件管理，该文件名为 listener.ora，该文件在 Linux 系统上默认位于$ORACL_HOME/network/admin 目录下。Oracle 的监听程序只运行在数据库服务器上，完成监听客户连接请求的作用。Oracle 使用 lsnrctl 实用程序完成监听程序的配置和管理。在第 2 章我们在 Linux 环境下创建数据库时已经启动了监听，使用默认端口 1521，此时会在 listener.ora 文件中出现如例子 7-1 所示的信息。

例子 7-1 监听配置信息

```
    #         listener.ora         Network         Configuration         File:
/u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
    # Generated by Oracle configuration tools.

    LISTENER =
      (DESCRIPTION_LIST =
        (DESCRIPTION =
          (ADDRESS = (PROTOCOL = TCP)(HOST = myoracle)(PORT = 1521))
        )
      )

    ADR_BASE_LISTENER = /u01/app/oracle
```

监听器 LISTNER 使用的协议为 TCP ，HOST 为 myoracle，端口是 1521（默认端口），该监听器为启动监听的默认监听器，即如果不具体指定启动哪个监听器，则默认启动名为 LISTNER 的监听器，当然也可以配置一个其他名字的监听器，端口也可以不同，如例子 7-2 所示，我们添加一个监听器 LISTENER1。

例子 7-2 增加监听器 LISTENER1

```
    LISTENER =
      (DESCRIPTION_LIST =
        (DESCRIPTION =
          (ADDRESS = (PROTOCOL = TCP)(HOST = myoracle)(PORT = 1521))
        )
```

```
  )

ADR_BASE_LISTENER = /u01/app/oracle

LISTENER1 =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = myoracle)(PORT = 1522))
    )
  )
```

Listener.ora 文件配置完成之后，我们使用 lsnrctl 指令启动监听，并查看监听状态，如例子 7-3 所示我们启动监听器 LISTNER1。

例子 7-3　启动监听器 LISTENER1

```
[oracle@myoracle admin]$ lsnrctl start listener1

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 17-DEC-2012 22:12:18

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Starting /u01/app/oracle/product/11.2.0/dbhome_1/bin/tnslsnr: please wait...

TNSLSNR for Linux: Version 11.2.0.1.0 - Production
System                  parameter                   file                    is
/u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
Log messages written to /u01/app/oracle/diag/tnslsnr/myoracle/listener1/alert/log.xml
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1522)))

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1522)))
STATUS of the LISTENER
------------------------
Alias                     listener1
Version                   TNSLSNR for Linux: Version 11.2.0.1.0 - Production
Start Date                17-DEC-2012 22:12:19
Uptime                    0 days 0 hr. 0 min. 0 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File  /u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
Listener Log File     /u01/app/oracle/diag/tnslsnr/myoracle/listener1/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1522)))
The listener supports no services
The command completed successfully
```



此时我们只启动了监听器 LISTENER1，但是没有任何服务注册到该监听器，监听器 LISTNER 也是如此。同样启动 LISTNER。

下面我们查看监听器 LISNTER1 的状态信息。看是否有服务注册过来。

例子 7-4　查看监听状态

```
[oracle@myoracle admin]$ lsnrctl status listener1

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 17-DEC-2012 22:13:12

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1522)))
STATUS of the LISTENER
------------------------
Alias                     listener1
Version                   TNSLSNR for Linux: Version 11.2.0.1.0 - Production
Start Date                17-DEC-2012 22:12:19
Uptime                    0 days 0 hr. 0 min. 54 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File  /u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
Listener Log File     /u01/app/oracle/diag/tnslsnr/myoracle/listener1/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1522)))
The listener supports no services
The command completed successfully
```

显然此时，没有任何服务注册在该服务器上，客户端无法连接到该服务器。此时我们没有涉及服务名的问题，而只有客户端将服务名传给监听器以后，监听器服务经过处理才能建立客户端到数据库服务器的连接。下面我们分别介绍动态注册和静态注册的概念。

7.2.1　动态注册

数据库实例里的 PMON 进程，将当前实例的服务名注册到同一台服务器上，默认在 1521 端口上监听的监听器就是名为 LISTENER 的监听器，当前实例的服务名由初始化参数 service_name 决定。我们首先查看当前数据库的服务名信息。

例子 7-5　查看当前数据库的服务名

```
SQL> show parameter service_name;

NAME                    TYPE         VALUE
------------------------------------ ------------------------------
service_names              string           orcl
```

显示当前的数据库服务器的 service_name 为 orcl。我们接下来手工完成动态注册，参数 local_listener 控制动态注册到的监听器。也就是将数据库动态注册到那个监听器。

我们查看当前该参数的值。

例子 7-6　查看当前数据库参数 local_listener 的值

```
SQL> show parameter local_listener;
```

```
NAME                    TYPE                        VALUE
------------------------------------ ----------------------------
local_listener                  string
```

我们发现，当前该参数的值为空，没有设置动态注册的监听器的信息。我们设计将数据库 orcl 动态注册到监听器 LISTENER1。下面在没有配置 local_listener 参数之前，我们再次确定监听器 LISTERER1 的状态信息。

例子 7-7　查看监听器 LISTERER1 的状态

```
[oracle@myoracle admin]$ lsnrctl status listener1

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 17-DEC-2012 22:17:52

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1522)))
STATUS of the LISTENER
------------------------
Alias                     listener1
Version                   TNSLSNR for Linux: Version 11.2.0.1.0 - Production
Start Date                17-DEC-2012 22:12:19
Uptime                    0 days 0 hr. 4 min. 34 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File  /u01/app/oracle/product/11.2.0/dbhome 1/network/admin/listener.ora
Listener Log File     /u01/app/oracle/diag/tnslsnr/myoracle/listener1/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1522)))
The listener supports no services
The command completed successfully
```

此时，依然没有任何服务注册过来。下面我们设置参数 local_listener，告诉数据库 orcl 要动态注册到监听器 LISTENER1，如例子 7-8 所示。

例子 7-8　修改参数 local_listener 动态注册 orcl 数据库

```
SQL> alter system set
  2
local listener='(address list=(address=(protocol=tcp)(host=23.9.1.100)(port=15
22)))';

System altered.
```

我们在 orcl 数据库上实现上述指令，告诉该数据库服务要动态注册到的监听器的具体信息，该参数一旦修改完毕，立即生效。

此时必须正确填写监听器 LISTENER 的信息，包括使用协议、监听器所在主机地址和使用的监听端口。因为是“动态注册”，所以我们不需要重启监听 LISTENER1，该服务 PROD 就自动注册到监听器 LISTENER1 上去。

下面我们查询 LISTENER1 的状态，如例子 7-9 所示。

例子 7-9　查看 LISTENER1 的状态

```
[oracle@myoracle admin]$ lsnrctl status listener1

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 17-DEC-2012 22:20:04

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1522)))
STATUS of the LISTENER
------------------------
Alias                     listener1
Version                   TNSLSNR for Linux: Version 11.2.0.1.0 - Production
Start Date                17-DEC-2012 22:12:19
Uptime                    0 days 0 hr. 7 min. 47 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File  /u01/app/oracle/product/11.2.0/dbhome 1/network/admin/listener.ora
Listener Log File     /u01/app/oracle/diag/tnslsnr/myoracle/listener1/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1522)))
Services Summary...
Service "orcl" has 1 instance(s).
  Instance "orcl", status READY, has 1 handler(s) for this service...
Service "orclXDB" has 1 instance(s).
  Instance "orcl", status READY, has 1 handler(s) for this service...
The command completed successfully
```

通过 Service Summary 我们知道，服务 orcl 已经注册到了监听器 LISTENER1。下面我们通过一个连接测试注册信息。

7.2.2　静态注册

静态注册是将数据库的信息直接注册到监听器配置文件中，这样只要监听启动就会静态注册该服务，这里“静态”要求必须重启监听器该注册才有效，这个概念类似 Oracle 的“静态”与“动态”参数的含义。静态与动态是相对于监听器的配置文件而言，如果在监听器的配置文件中注册数据库则是静态注册。下面是一个静态注册的例子，我们将数据库 orcl 注册到默认监听器 LISTENER。此时需要修改 listener.ora 文件，修改后的文件内容如例子 7-10 所示。

例子 7-10　静态注册 orcl 数据库到 LISTENER 监听器

```
LISTENER =
  (DESCRIPTION LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = myoracle)(PORT = 1521))
    )
  )

SID LIST LISTENER=
   (SID_LIST=
```

```
        (SID DESC=
            (SID_NAME=orcl)
            (ORACLE_HOME=/u01/app/oracle/product/11.2.0/dbhome_1)
            (GLOBAL DBNAME=orcl)
        )
      )

ADR BASE LISTENER = /u01/app/oracle

LISTENER1 =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = myoracle)(PORT = 1522))
    )
  )
```

我们将数据库 PROD 注册到了监听器 LISTENER，此时使用静态注册直接在 listener.ora 文件中注册该数据库。这里需要 3 个参数 SID_NAME、ORACLE_HOME 和 GLOBAL_DBNAME。

完成静态注册的内容修改后，我们暂时不重启监听器 LISTENER。看数据库 orcl 服务是否注册到了监听器 LISTENER。如例子 7-11 所示。

例子 7-11　查看监听器 LISTENER 的状态

```
[oracle@myoracle admin]$ lsnrctl status listener

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 17-DEC-2012 22:34:42

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1521)))
STATUS of the LISTENER
------------------------
Alias                     listener
Version                   TNSLSNR for Linux: Version 11.2.0.1.0 - Production
Start Date                17-DEC-2012 22:28:03
Uptime                    0 days 0 hr. 7 min. 39 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
Listener Log File         /u01/app/oracle/diag/tnslsnr/myoracle/listener/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1521)))
The listener supports no services
The command completed successfully
```

显然，没有任何服务注册过来，下面我们重启该监听。先关闭 LISTENER，而后启动监听，此时新的监听文件内容生效。

例子 7-12　重新启动监听 LISTENER

```
[oracle@myoracle admin]$ lsnrctl stop listener
```

```
LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 17-DEC-2012 22:27:57

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1521)))
The command completed successfully
[oracle@myoracle admin]$ lsnrctl start listener

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 17-DEC-2012 22:28:02

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Starting /u01/app/oracle/product/11.2.0/dbhome_1/bin/tnslsnr: please wait...

TNSLSNR for Linux: Version 11.2.0.1.0 - Production
System parameter file is /u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
Log messages written to /u01/app/oracle/diag/tnslsnr/myoracle/listener/alert/log.xml
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1521)))

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1521)))
STATUS of the LISTENER
------------------------
Alias                     listener
Version                   TNSLSNR for Linux: Version 11.2.0.1.0 - Production
Start Date                17-DEC-2012 22:28:03
Uptime                    0 days 0 hr. 0 min. 0 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
Listener Log File         /u01/app/oracle/diag/tnslsnr/myoracle/listener/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1521)))
Services Summary...
Service "orcl" has 1 instance(s).
  Instance "orcl", status UNKNOWN, has 1 handler(s) for this service...
The command completed successfully
```

在重启监听的过程中，输出显示服务 orcl 已经注册进来，不过为了规范我们使用 status 关键字查询数据库服务 orcl 是否静态注册到监听器 LISTENER。

例子 7-13　查询监听器 LISTENER 的状态

```
[oracle@myoracle admin]$ lsnrctl status listener

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 17-DEC-2012 22:28:13

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1521)))
```

```
STATUS of the LISTENER
------------------------
Alias                     listener
Version                   TNSLSNR for Linux: Version 11.2.0.1.0 - Production
Start Date                17-DEC-2012 22:28:03
Uptime                    0 days 0 hr. 0 min. 10 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File  /u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
Listener Log File        /u01/app/oracle/diag/tnslsnr/myoracle/listener/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1521)))
Services Summary...
Service "orcl" has 1 instance(s).
  Instance "orcl", status UNKNOWN, has 1 handler(s) for this service...
The command completed successfully
```

显然，从 Services Summary...知道数据库服务 orcl 静态注册到了监听器 LISTENER。

7.2.3　连接测试

当我们使用 SQL*Plus 从客户端连接到数据库服务器时，虽然监听已经启动，并且我们完成了静态注册和动态注册，但是作为客户端还必须有自己的配置，这里我们不详细介绍，只是给出一个例子，演示我们监听器静态和动态注册的结果。

在客户端需要修改 tnsnames.ora 文件时，将客户端的服务名与监听器服务名之间建立映射关系，并告诉客户端软件要连接到数据库服务器的实体信息，这些信息包括通信协议、数据库服务器主机地址以及使用连接端口等。该文件与 listener.ora 文件位于同一个目录，在开始时该文件不存在，需要自己创建。下面是 tnsnames.ora 文件的配置信息。

例子 7-14　配置 tnsnames.ora 文件

```
myorcl =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = myoracle)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )
```

下面我们测试客户端到监听器是否畅通，使用 tnsping 指令，如例子 7-15 所示。

例子 7-15　测试客户端到监听器是否畅通

```
[oracle@myoracle admin]$ tnsping myorcl;

TNS Ping Utility for Linux: Version 11.2.0.1.0 - Production on 17-DEC-2012
22:47:37
```

```
Copyright (c) 1997, 2013, Oracle.  All rights reserved.

Used parameter files:


Used TNSNAMES adapter to resolve the alias
Attempting to contact (DESCRIPTION = (ADDRESS LIST = (ADDRESS = (PROTOCOL =
 TCP)(HOST = myoracle)(PORT = 1521))) (CONNECT DATA = (SERVER = DEDICATED)
(SERVICE NAME = orcl)))
OK (50 msec)
```

显然此时客户端到监听器之间是畅通的，但是还不能保证到数据库 orcl 的连接一定成功，因为如果数据库没有启动，而只是启动了监听，此时一样可以 Ping 通。接下来我们测试到数据库服务器的连接是否成功，我们使用 sqlplus 指令。如例子 7-16 所示。

例子 7-16　测试到数据库服务器的连接

```
[oracle@myoracle ~]$ sqlplus system/Oracle1234@myorcl

SQL*Plus: Release 11.2.0.1.0 Production on Thu Dec 20 11:18:50 2012

Copyright (c) 1982, 2013, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

显然，此时连接成功，客户端成功建立到数据库 PROD 的连接。接下来我们将详细介绍客户端的配置以及几种客户端连接数据库的方法。下面我们通过一个具体的动态性能视图确认该连接的存在和相关进程信息。

例子 7-17　确认网络简易连接方式

```
SQL> select sid ,serial#,username,program,machine,status,port from v$session
  2* where username='SYSTEM'

  SID    SERIAL# USERNAME   PROGRAM                  MACHINE    STATUS
---------- ------------------------ ----------------- ----------------
  33      92       SYSTEM     sqlplus@myoracle (TNS V1-V3)   myoracle     ACTIVE
```

默认情况下，SYSTEM 用户是锁定（Lock）的，需要先解锁方可使用。可以用如例子 7-18 所示的指令解锁。

例子 7-18　解锁用户

```
SQL> alter user scott  account unlock identified by Oracle1234;

User altered.
```

7.2.4　监听程序管理

Oracle 使用 lsnrctl 实用程序来管理和维护监听，如使用 status 指令查询监听状态信息，通过 service 指令查看监听为连接请求监控的服务内容等。而更多的指令，我们可以通过 help 指令查看。如例子 7-19 所示。

例子 7-19　查看监听管理指令

```
[oracle@myoracle ~]$ lsnrctl help

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 20-DEC-2012 11:29:57

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

The following operations are available
An asterisk (*) denotes a modifier or extended command:

start               stop                status
services             version             reload
save config           trace              spawn
change password        quit               exit
set*                show*
```

下面解释常用的几个指令。

- Lsnrctl start：启动监听程序，默认启动名为 LISTENER 的监听程序，如果不是默认监听则需要制定启动的监听器名字，如 lsnrctl start lsnr2。
- Lsnrctl stop：关闭监听程序，默认启动名为 LISTENER 的监听程序，如果不是默认监听则需要制定关闭的监听器名字，如 lsnrctl stop lsnr2。
- Lsnrctl service：查看监听为连接请求监控的服务内容。
- Lsnrctl reload：允许重载监听程序，如监听文件更改，可以使用该指令使得修改生效，在重载过程中已经建立连接的客户机继续保持连接。
- Lsnrctl set：设置监听管理密码，防止未授权而操作监听器。

下面我们给出设置监听程序管理的几个例子。

（1）多监听设置，在一个服务器上可以运行多个监听程序，在 RAC 环境下比较普遍，此时往往需要设置 CONNECT_TIME_FAILOVER 参数，其含义是当客户机使用新的监听程序建立连接前需要等待通过当前监听程序连接的时间长度。

（2）设置队列长度。如果有大量客户请求连接往往使得监听程序无法提供足够的连接资源，此时发生监听程序失败难以避免，为了控制客户连接到监听器的数量，在监听器文件中设置 queuesize 参数，告诉监听器可以为多少个连接提供并发服务。

```
LISTENER=
   (DESCRIPTION LIST=
      (DESCRIPTION=
        (ADDRESS=(PROTOCOL=TCP)(HOST=ocm1.oracle)(PORT=1521) (QUEUESIZE=5)
      )
```

```
    )
```

（3）设置监听程序密码。因为在设置监听程序时，默认是没有保护密码的，这就对监听的管理带来混乱，因为任何进入操作系统的人都可以操作监听器，这样对于安全管理显然不利，如例子7-20所示。

例子 7-20　设置监听密码

```
LSNRCTL> set password
Password:
The command completed successfully
LSNRCTL> change password
Old password:
New password:
Reenter new password:
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1521)))
Password changed for LISTENER
The command completed successfully
LSNRCTL> save_config
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1521)))
Saved LISTENER configuration parameters.
Listener Parameter File  /u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
Old Parameter File  /u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.bak
The command completed successfully
```

在启动和关闭监听时，就需要使用密码来完成，显然增强了监听管理的安全性。

7.3　客户端配置

客户端要连接到数据库服务器必须知道一些信息，如数据库服务器的 ip 地址，使用的通信协议、连接端口号，以及对应的数据库服务名，这些需要在数据库客户端设置，这些信息存储在 tnsnames.ora 文件中，在该文件中设置要连接到的每一个数据库服务器的实体信息。

7.3.1　本地命名

本地命名使用的配置文件是 tnsnames.ora 文件，该文件位于客户端。我们可以使用 netca 来配置这个文件。本地命名是建立 Oracle 数据库连接的最便捷的方法，在文件 tnsnames.ora 中存储了数据库服务名和链接描述符，该文件默认位于$ORACLE_HOME/network/admin 目录中。

在使用 SQL*Plus 或其他方法初始化一个客户端连接时，Oracle Net 组件会提供数据库详细信息，这些信息就存储在 tnsnames.ora 文件中，这些信息包括数据库服务器的网络地址、通信协议以及连接端口。通过这些信息与监听程序建立通信，而后监听程序会将该连接交给 SERVICE_NAME 指定的数据库，一旦数据库服务器确定了用户名和密码匹配则建立客户端的连接。此时的 tnsnames.ora 就是一个服务名解析的作用，将一个服务名解析为一个网络连接实体，确定要连接的具体的数据库服务。

下面是 tnsnames.ora 文件的内容，其中我们添加了数据库 ORCL 的服务名信息。如果客户端需要连接到更多的数据库，则需要添加更多的映射信息。

```
MYORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = myoracle)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )
```

ADDRESS 是监听器的实体信息，CONNECT_DATA 说明使用的服务器体系架构和对应的数据库服务名。这样客户端软件就可以根据这些信息建立到监听器的连接，而监听器根据 SERVICE_NAME 找到注册的数据库，将连接交给数据库服务器，一旦数据库服务器通过用户名和密码验证，则建立客户端和服务器之间的连接。

tnsnames.ora 文件必须位于客户端上，如果有多个客户端则需要复制过去，保证连接成功，要求数据库服务器端的监听程序必须运行，数据库服务器保持运行。

下面我们通过如下语法测试连接。

```
SQL>connect username/password@net_servive_name
```

例子 7-21　测试本地连接

```
[oracle@myoracle ~]$ sqlplus system/Oracle1234@myorcl

SQL*Plus: Release 11.2.0.1.0 Production on Thu Dec 20 11:49:21 2012

Copyright (c) 1982, 2013, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

7.3.2　简单命名

简单命名网络连接是最简洁的连接方法，在配置方面要求很少。此时客户端用户不需要配置 TCP/IP 环境下的 tnsnames.ora 文件就可以连接到 Oracle 数据库服务器。客户机只需要主机名（或者 IP 地址）、端口号以及数据库的服务名即可连接数据库服务器，这样就省去了系统中对数据库的 TCP/IP 连接实体配置。

但是使用简单连接要求在客户机以及数据库服务器上同时运行 TCP/IP 协议栈，简单连接在 Oracle 9i 中引入，是数据库连接的有益补充。下面是简单连接的语法。

```
$CONNECT username/password@[//]host[:port][/service_name]
```

下面解释下简单连接中的几个注意事项。

- //（双斜杠）是可选的。
- Host 是强制参数，可以是主机名也可以是 IP 地址。
- Port 是端口号，该参数是可选的，如果是默认端口可以不写。
- Service_name 参数指定数据库服务器的服务名，默认该主机名是不选参数。如果主机名和数据库服务器名相同，则这个参数可以省略；如果不同，则必须提供数据库服务器的服务名。

下面是一个连接实例。

例子 7-22　测试简单连接方式

```
[oracle@myoracle ~]$ sqlplus system/Oracle1234@myoracle:1521/orcl

SQL*Plus: Release 11.2.0.1.0 Production on Thu Dec 20 11:47:38 2012

Copyright (c) 1982, 2013, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

数据库服务器所在的 IP 地址为 myorcl，使用默认端口 1521 作为监听端口，数据库服务器的服务名为 orcl。

既然是默认端口号，此时可以不写这个参数。实质上使用简单连接方法与使用本地命名方法是相同的，在简单连接方法中提供的连接参数，在本地命名连接方法中的 tnsnames.ora 文件中都有，如下所示。

```
MYORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = myoracle)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )
```

7.4　Oracle数据库服务器支持的两种连接方式

Oracle 提供了两种数据库连接方式，一种是专有连接方式，一种是共享连接方式，二者的区别就是对服务器进程的拥有模式，专有连接方式一个用户连接对应一个数据库服务器进程，而共享连接方式多个用户可以使用一个数据库服务器（严格意义上讲是分时复用），使用共享服务器模式对于事务执行时间短且服务器资源受限的系统是有利的，但是 Oracle 依然推荐使用专有连接方式。这种方式会耗费内存资源（PGA），但是会减少竞争，对于长事务尤其有用。

7.4.1　服务器进程

Oracle 提供了两种网络连接方式，一种是专有连接，一种是共享连接，面对这两种连接方式Oracle 对应两种数据库服务器：专有服务器和共享服务器。无论是那种连接方式，都需要服务器进程的参与，服务器进程是代表客户完成数据库访问工作的进程，客户端的 SQL 语句都通过这些服务器进程来接受并严格执行。

- 专有服务器：采用专有服务器时，数据库服务器会针对每一个连接产生一个服务器进程，这个进程就是专属这个连接的。数据库连接于服务器上的一个进程或线程之间是一一对应关系。
- 共享服务器：采用共享服务器时，数据库服务器的一个服务器进程服务于多个会话，即使用服务器进程池（有多个服务器进程）为多个会话服务，在建立连接时，客户端会首先连接到调度器，由调度器协调数据库服务器的资源。

7.4.2　共享连接

共享服务器模式下，用户进程与服务器进程是多对多的关系，多个服务器进程会处理多个用户进程。在使用共享服务器连接时，必须使用 Oracle Net 协议软件，如果不使用 Oracle 监听器，则无法使用共享服务器。

整个共享连接的过程是客户发起连接，监听器处理连接并将连接重定向或者转交给一个调度器，调度器在协调再享服务器资源，如图 7-1 所示。

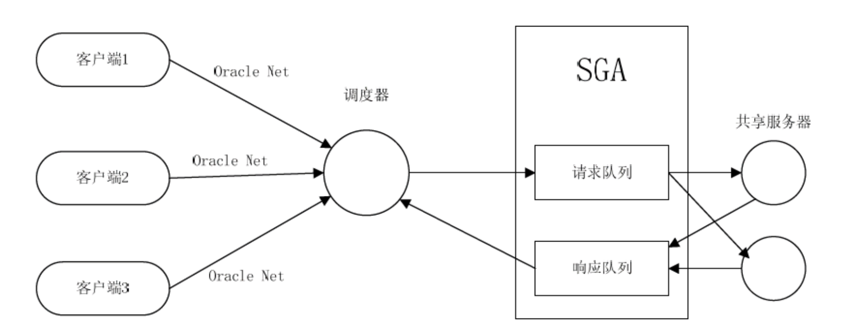


图 7-1　共享服务器连接示意图

在图 7-1 中，多个客户端通过网络发起连接，目标是连接到 Oracle 数据库，此时客户应用程序会与一个调度器 DISPATCHER 建立物理连接。调度器可以配置多个，也可以只配置一个，因为调度器的任务不是很重，它将负责接受客户请求，并将客户请求放入 SGA 的请求队列，然后调度器会不断地监视响应队列，一旦查询结果返回则把结果传回给用户。

共享服务器进程一旦空闲，则从请求队列获取用户请求，处理这个请求，将数据放回响应队列。而对于客户而言，这些操作都是透明的，无论使用专有服务器连接还是共享服务器连接，其实对于客户而言感受不到不同服务器架构的存在，只是在特定的数据库应用环境下，需要考虑专有服务器连接和共享服务器连接的优劣，选择合适的网络连接方式。

7.4.3 共享连接涉及初始化参数

共享服务器中涉及很多参数，如调度器、共享服务器、会话等，本节我们讨论这些参数的使用以及含义。

- 调度器（DISPATCHERS）：调度器负责将用户请求传递放入 SGA，并负责监控 SGA 中的响应队列，如果有返回数据则将数据返回给用户。
- 共享服务器进程（Shared_servers）：该参数设置共享服务器进程的数量。
- 最大共享服务器进程（Max_shared_servers）：该参数设置数据库服务器支持的最多的服务器进程数量。
- 共享服务器会话数（shared_server_sessions）：该参数设置当前数据库服务器在共享模式下的最大会话数。而专用会话数等于 sessions-共享会话数。

7.4.4 共享连接的工作过程

共享连接模式下，需要配置多个名为 DISPATCHER 的组件，DISPATCHER 作为用户进程和服务器进程之间的协调者，负责将用户进程的请求传递给服务器进程，并将服务器进程得到的结果返回给用户。

PMON 定期将每个 DISPATCHER 的地址，以及工作负载注册到监听器里面，当用户进程连接监听器时，监听器会选择一个负载最低的 DISPATCHER，并把该 DISPATCHER 的地址返回给用户进程，用户进程和 DISPATCHER 进程建立连接，用户进程在 SESSION 的整个生命期间，所连接的 DISPATCHER 不会发生改变。

整个连接如图 7-2 所示。

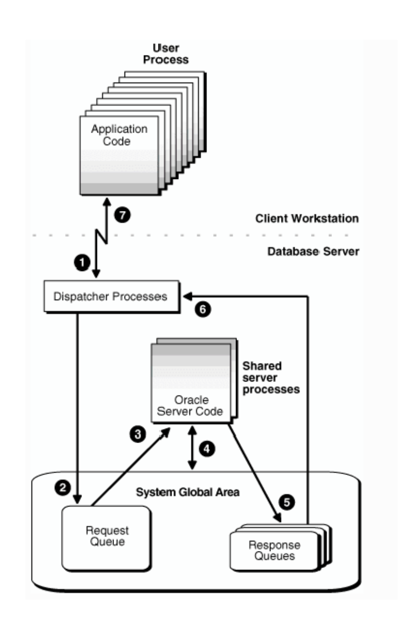


图 7-2 共享连接流程图

- 用户进程连接到监听器。
- 监听器根据注册的各个 DISPATCHER 的负载情况，选择一个负载最低的 DISPATCHER，并将其地址返回给用户进程。
- 用户进程根据监听器返回的 DISPATCHER 地址，连接到该 DISPATCHER。
- DISPATCHER 接收到用户进程发出的请求以后，会将该请求放入请求队列，该队列位于 SGA 中，请求队列被所有的 DISPATCHER 共有。
- 在服务器进程中，最空闲的服务器进程会从请求队列中按照先进先出的原则，挑选一个请求进行处理。
- 服务器进程处理请求后，得到的结果放入响应队列，Oracle 为每个 DISPATCHER 分配一个对应的响应队列。
- DISPATCHER 到相应的队列中取出结果，返回给用户进程。

共享连接中，多个服务器进程会处理多个用户请求，用户的 PGA 就需要在不同的服务器进程之间共享，PGA 中的 UGA 部分就会被放入到 SGA 中，如果配置了 Large pool，则 UGA 会在 Large pool 里分配，没有配置 Large pool，那么 UGA 就放在 Shared pool 里面。

7.4.5　共享连接的配置

共享连接涉及如下参数：dispatchers、shared_servers、max_shared_servers、shared_server_sessions。下面我们依次介绍并解释其作用。

1. 配置 DISPATCHER 进程的数量

DISPATCHERS 初始化参数是用来配置 DISPATCHERS 进程信息，如连接使用协议，DISPATCHERS 进程数量，使用端口等。下面是典型的 DISPATCHERS 配置。

例子 7-23　配置 DISPATCHERS

```
SQL> alter system set dispatchers='(protocol=tcp)(dispatchers=3)';

System altered.
```

对于 DISPATCHERS 进程有 3 类属性设置需要考虑（有的是可选的）。

协议地址，该属性由 3 个参数设置。

- ADDRESS：DISPATCHERS 监听所在网络地址。
- DESCRIPTION：DISPATCHERS 监听所在网络地址描述，包括网络协议地址。
- PROTOCOL：指明 DISPATCHERS 使用的网络协议。

指定 DISPATCHERS 的数量。在设置 DISPATCHERS 进程信息时，也可以指定在哪个地址上创建监听，如下所示。

```
DISPATCHERS='(ADDRESS=(PROTOCOL=TCP)(HOST=144.25.17.201))(DISPATCHERS=2)'
```

在设置该参数时，需要考虑 max_dispatchers 参数，这个参数限制了 DISPATCHERS 的最大数量。如下所示，我们限制这个最大数值为 5。

例子 7-24　修改参数

```
SQL> alter system set max dispatchers=5;

System altered.
```

2. 配置共享服务器进程的数量

参数 shared_servers 设置当前数据库服务器启动的共享服务器数量，而参数 max_shared_servers 设置当前数据库服务器支持的最大共享服务器数量。下面我们修改这两个值。

例子 7-25　修改参数

```
SQL> alter system set shared servers=5;

System altered.

SQL> alter system set max shared servers=20;

System altered.

Shared_servers , max_shared_servers。
```

我们设置当前共享数据库服务器进程数量为 5，而最大共享服务器数为 20；服务器进程动态调整，首先是分配默认的数量，根据负载适当地增加，当负载下降的时候，适当减少服务器进程的数量。但是增加和减少的数量控制在上面的两个参数之间。

3. 配置共享会话数

我们需要设置 Oracle 分配给共享会话的连接数量，设置 shared_server_sessions 参数，该参数的设置与 sessions 参数有关 ，sessions 参数指定数据库最大的会话数，那么共享会话数就不能超过 sessions 参数指定的值。而专有连接数量等于 sessions 减去共享连接数。

设置数据库的共享会话数为 100，如下例所示。

例子 7-26　修改共享会话数

```
SQL> alter system set shared_servers_sessions=100;

System altered.
```

接着我们查看当前的最大会话数是多少。

例子 7-27　查看最大会话数

```
SQL> show parameter sessions;

NAME                             TYPE        VALUE
------------------------ ----------- ------------------------------
java_max_sessionspace_size       integer     0
java soft sessionspace limit     integer     0
license max sessions             integer     0
license sessions warning         integer     0
sessions                         integer     247
shared_server_sessions           integer     100
```

此时显示当前的最大会话数是 225，共享连接会话数是 100，那么专有连接会话数就是 147。

4．修改 tnsnames.ora 文件

修改如下所示。

```
MYORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = myoracle)(PORT = 1522))
    )
    (CONNECT_DATA =
      (SERVER = SHARED)
      (SERVICE_NAME = orcl)
    )
  )
```

此时，我们可以使用本地连接方式，数据库动态注册到监听器 LISTENER1，所以此时的 PORT 为 1522，要实现动态注册还需要设置 local_listener 参数，这个参数我们在 7.2.1 节已经设置过了。我们需要启动 LISTENER1。然后查看该监听支持的服务。

例子 7-28　启动监听 LISTENER1

```
[oracle@myoracle admin]$ lsnrctl start listener1

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 20-DEC-2012 21:23:45

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Starting /u01/app/oracle/product/11.2.0/dbhome_1/bin/tnslsnr: please wait...

TNSLSNR for Linux: Version 11.2.0.1.0 - Production
System parameter file is /u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
Log messages written to /u01/app/oracle/diag/tnslsnr/myoracle/listener1/
alert/log.xml
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1522)))

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1522)))
STATUS of the LISTENER
------------------------
Alias                     listener1
Version                   TNSLSNR for Linux: Version 11.2.0.1.0 - Production
Start Date                20-DEC-2012 21:23:45
Uptime                    0 days 0 hr. 0 min. 0 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /u01/app/oracle/product/11.2.0/dbhome_1/network/admin/listener.ora
Listener Log File         /u01/app/oracle/diag/tnslsnr/myoracle/listener1/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=1522)))
The listener supports no services
The command completed successfully
```

上面我们启动了监听器 LISTENER1，然后我们就更可以通过该监听器实现数据库的动态注册。下面通过本地命名方式连接数据库。

例子 7-29　本地命名方式连接数据库

```
[oracle@myoracle ~]$ sqlplus system/Oracle1234@myorcl

SQL*Plus: Release 11.2.0.1.0 Production on Thu Dec 20 21:18:34 2012

Copyright (c) 1982, 2013, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

此时，通过本地连接方式连接到了数据库服务器，使用的是共享连接。我们接下来继续查看监听器 LISTENER1 的服务信息，确认是共享连接。

例子 7-30　查看监听器 LISTENER1 的服务信息

```
[oracle@myoracle admin]$ lsnrctl service listener1

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 20-DEC-2012 21:19:50

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Connecting                                                                  to
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1522)))
Services Summary...
Service "orcl" has 1 instance(s).
  Instance "orcl", status READY, has 4 handler(s) for this service...
    Handler(s):
      "D002" established:0 refused:0 current:0 max:1022 state:ready
         DISPATCHER <machine: myoracle, pid: 11359>
         (ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=34253))
      "D001" established:0 refused:0 current:0 max:1022 state:ready
         DISPATCHER <machine: myoracle, pid: 11357>
         (ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=34254))
      "D000" established:1 refused:0 current:1 max:1022 state:ready
         DISPATCHER <machine: myoracle, pid: 7575>
         (ADDRESS=(PROTOCOL=tcp)(HOST=myoracle)(PORT=32810))
      "DEDICATED" established:0 refused:0 state:ready
         LOCAL SERVER
  Service "orclXDB" has 1 instance(s).
  Instance "orcl", status READY, has 0 handler(s) for this service...
  The command completed successfully
```

上面输出结果表明使用了该 DISPATCHERS 建立的共享连接。D00、D001、D002 代表 3 个调度器 Dispatcher。

7.4.6　共享连接的一些问题

（1）有些操作不能使用共享连接

- 启动关闭数据库实例
- 创建表空间和数据文件
- 维护表和索引等数据库的管理工作

（2）有些操作不适合共享服务器连接

共享服务器适合单纯的 OLTP，对于需要扫描大量数据，运行时间较长的操作，不适合采用共享连接，例如备份恢复。

7.4.7　专有连接

在专有服务器模式下，客户连接与数据库服务进程之间是一对一的关系，一个客户连接对应一个服务器进程。

专用连接中，用户进程没有发出命令，服务器进程处于空闲状态，资源一直占用，共享模式中，只要是服务器进程空闲，就可以处理其他用户发出的命令，因此服务器进程的数量减少，对资源的利用更加高效，占用的 PGA 减少，可以支持更多的用户。这是共享服务器的优势，相反就是专有服务器的劣势了。

专有连接其实就是一个典型的客户——服务器 2 层架构。在应用程序中，客户应用程序通过 API 连接到数据库，而这些 API 知道如何将 SQL 查询传递给数据库，并处理返回的数据。这些 API 知道如何将数据库请求打包为网络调用，而专有服务器进程则知道如何解包这些网络应用。整个过程由一个网络协议软件或 Oracle Net 来支持。

图 7-3 是典型的专有服务器连接示意图。

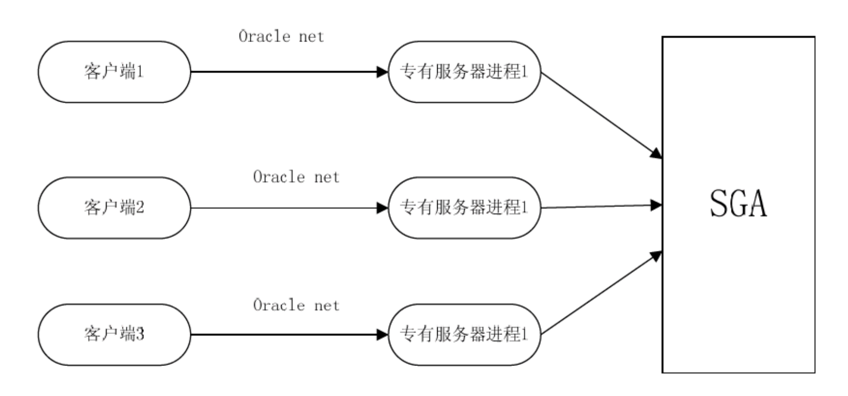


图 7-3　专有服务器连接示意图

在使用本地命名连接数据库时，使用默认专有连接也可以显示设置专有连接参数，如下所示为 tnsnames.ora 文件的内容。

```
MYORCL =
  (DESCRIPTION =
```

```
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = myoracle)(PORT = 1521))
      )
      (CONNECT_DATA =
        (SERVER = DEDICATED)
        (SERVICE_NAME = orcl)
      )
    )
```

此时客户端连接数据库服务器时，就使用专有连接方式，监听器 LISTENER 负责接收该客户端的连接请求，我们查看这个专有连接的建立后，监听器记录的信息。

例子 7-31　查看监听器 LISTENER 的服务信息

```
[oracle@myoracle admin]$ lsnrctl service listener

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 20-DEC-2012 21:40:27

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myoracle)(PORT=1521)))
Services Summary...
Service "orcl" has 1 instance(s).
  Instance "orcl", status UNKNOWN, has 1 handler(s) for this service...
    Handler(s):
      "DEDICATED" established:1 refused:0    //建立了专有连接
         LOCAL SERVER
The command completed successfully
```

7.5　数据库驻留连接池

DRCP 是数据库驻留连接池的意思，它是 Oracle 11g 中提供的一种新的数据库连接方式。它的最大优势是减少内存使用，但是同样的内存资源支持更多的用户。

在 Web 应用中，存在这样一种情况，即用户获得一个连接，很短的时间后又放弃该连接。所以，Web 应用不保持连续的数据库活跃连接。而使用数据库连接池将有助于通过少量的池服务器为成千上万的用户提供服务。DRCP 类似于专有服务器连接，但是用户会话不会一直占有某个服务器进程，在很短的时间内使用完毕后就释放池服务器，该池服务器回到连接池继续为其他会话服务。

7.5.1　DRCP 的工作原理

DRCP 数据库驻留连接池使用一个名为 BROKER 的中介，当用户连接到来时，它负责分配一个可用的池服务器。即在客户连接请求被数据库服务器批准后，在服务器池中将分配一个池服务器，分配给该用户连接使用，如果池中没有空闲池服务器，只要没有超过池服务器的最大值，BROKER 就可以创建新的池服务器。如果达到最大池服务器数目，则 BROKER 将用户连接放入请求队列，直到有空闲池服务器为止。显然使用 DRCP 使用的内存量只与活跃的池服务器数量有关，而与用户连接数量无关。这是 DRCP 与专有连接和共享连接相比最明显的优势，即节约资源。

DRCP 实际上专门针对 Web 流量创建了数据库驻留连接池（DRCP），它使用专用服务器和连接代理的组合来处理来自 Web 应用程序的简短、临时的会话。它通过一组超时、池限制和会话设置通过启动少数专用进程来处理进入池中的连接，从而解决了服务器资源耗尽的问题。

下面我们通过数据字典 dba_cpool_info 查询默认情况下连接池的信息。

例子 7-32　查询连接池信息

```
SQL> select connection pool,status,maxsize from dba cpool info

CONNECTION_POOL                STATUS           MAXSIZE
-------------------------------------------------------------------------
SYS_DEFAULT_CONNECTION_POOL    INACTIVE                40
```

从输出知道，系统的默认连接池处于禁用状态，要启动连接池功能需要使用 DBMS_CONNECTION_POOL 包的 START_POOL 过程，如下所示。

```
SQL> exec dbms_connection_pool.start_pool();
PL/SQL procedure successfully completed.
```

下面查询当前连接池的状态。

```
SQL> select connection_pool,status,maxsize from dba_cpool_info;

CONNECTION POOL              STATUS           MAXSIZE
-------------------------------------------------- ------------------
SYS_DEFAULT_CONNECTION_POOL    ACTIVE                  40
```

显然从输出知道，此时已经成功启动连接池，如果想禁止连接池，使用 STOP_POOL 方法，如下所示。

```
SQL> exec dbms_connection_pool.stop_pool();
PL/SQL procedure successfully completed.
```

既然有了一个新的功能，自然对应一个后台进程管理连接池，后台进程 CMON 管理连接池，负责将池服务器放回连接池。当我们使用 TNSNAMES.ORA 文件时，则在 TNS 连接字符串中必须指定 SERVER 参数为 POOLED。

7.5.2　如何配置 DRCP

连接池提供几个参数用于灵活配置实现系统需要，通过包 DBA_CONNECTION_POOL 的过程实现修改或者设置，过程 CONFIGURE_POOL 设置参数值，过程 ALTER_PARAM 修改参数。我们先介绍 DRCP 的几个参数。

- INACTIVITY_TIMEOUT：池服务器禁止服务前的最大空闲时间。
- MAX_LIFETIME_SESSION：每个池会话的存活时间。
- MAX_USE_SESSION：一个池服务器放到连接池的最大次数。
- MAXSIZE 和 MINSIZE：池服务器最大和最小个数。
- MAX_THINK_TIME：用户获得池服务器后保持不活动的最大时间。

这些参数可以通过 dbms_connection_pool 包中的过程来设置。过程 CONFIGURE_POOL 定义如下所示。

```
PROCEDURE CONFIGURE POOL
 Argument Name                  Type                    In/Out     Default?
 ------------------------------ ----------------------- ------ --------
 POOL_NAME                      VARCHAR2                IN         DEFAULT
 MINSIZE                        BINARY INTEGER          IN         DEFAULT
 MAXSIZE                        BINARY INTEGER          IN         DEFAULT
 INCRSIZE                       BINARY INTEGER          IN         DEFAULT
 SESSION_CACHED_CURSORS         BINARY_INTEGER          IN         DEFAULT
 INACTIVITY_TIMEOUT             BINARY_INTEGER          IN         DEFAULT
 MAX THINK TIME                 BINARY INTEGER          IN         DEFAULT
 MAX USE SESSION                BINARY INTEGER          IN         DEFAULT
 MAX_LIFETIME_SESSION           BINARY_INTEGER          IN         DEFAULT
```

下面查询当前默认连接池的相关参数的值。

例子 7-33　查看默认连接池的状态以及相关参数

```
SQL> col connection pool for a30
SQL> set line 120;
SQL> select connection_pool,status,maxsize,minsize,inactivity_timeout
  2* from dba cpool info

CONNECTION POOL        STATUS     MAXSIZE  MINSIZE  INACTIVITY TIMEOUT
------------------------------ -----------------------------------------
SYS_DEFAULT_CONNECTION_POOL  ACTIVE        40         4       300
```

下面我们演示如何设置池服务器的最大数，我们知道池服务器默认最大为 40 个，下面修改为 70 个。

```
SQL> exec dbms connection pool.alter param('','maxsize','70');

PL/SQL procedure successfully completed.
```

下面我们验证上述修改结果。

```
SQL> col connection pool for a30
SQL> select connection_pool,status,maxsize from dba_cpool_info

CONNECTION POOL              STATUS            MAXSIZE
------------------------------------------------ ----------------------
SYS_DEFAULT_CONNECTION_POOL    ACTIVE                  70
```

通过输出知道，此时的池服务器的最大数量是 70，说明修改成功。

下面修改客户端的 TNSNAMES.ORA 文件，以使得该客户端的连接使用 DRCP 连接池功能。

```
MYORCL =
  (DESCRIPTION =
    (ADDRESS LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = myoracle)(PORT = 1522))
    )
    (CONNECT_DATA =
```

```
        (SERVER = POOLED)
        (SERVICE_NAME = orcl)
      )
    )
```

上面主要是修改 SERVER 参数为 POOLED，并且使用实现动态注册的监听器 LISTENER1（通过端口号区别）。在设置完后，我们启动监听器 LISTENER1。然后连接数据库，此时使用不同用户多连接几次，测试池服务器分配情况以及池服务器请求数量。

我们首先使用简单连接方式测试使用 DRCP 到数据库服务器的连接。

例子 7-34　使用简单连接方式连接数据库服务器

```
[oracle@myoracle admin]$ sqlplus
system/Oracle1234@myoracle:1522/orcl:pooled;

SQL*Plus: Release 11.2.0.1.0 Production on Mon Dec 24 21:47:37 2012

Copyright (c) 1982, 2013, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

下面我们使用本地命名方式连接数据库服务器。

```
 [oracle@myoracle ~]$ sqlplus system/Oracle1234@myorcl

SQL*Plus: Release 11.2.0.1.0 Production on Thu Dec 20 23:18:01 2012

Copyright (c) 1982, 2013, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

然后通过数据字典视图 v$cpool_stats 查询池服务器的请求信息。

例子 7-35　查看池服务器状态信息

```
SQL>select pool_name,num_open_servers,num_requests,num_hits,num_misses
  2* from v$cpool_stats

POOL_NAME        NUM_OPEN_SERVERS NUM_REQUESTS NUM_HITS NUM_MISSES
---------------------------- ---------- ------------- ------------------
SYS_DEFAULT_CONNECTION_POOL  4           3               0        3
```

下面解释数据字典 v$cpool_stats 中几个重要列的含义。

- NUM_OPEN_SERVERS：服务器池中空闲和被占用的服务器数量。
- NUM_REQUESTS：客户端请求服务器池中的服务器的次数。
- NUM_HITS：客户端请求池中的服务器且获得服务器的次数。
- NUM_MISSES：客户端请求池中的服务器而没有获得服务器的次数。

下面再通过动态性能视图 v$cpool_conn_info 获取通过服务器池连接到数据库服务器的用户和进程信息。

例子 7-36　查询 DRCP 下用户和进程信息

```
SQL>select session addr,username,process id,connection status
  2* from v$cpool conn info

SESSION_    USERNAME           PROCESS_ID       CONNECTION
-----------------------------------------------------------------------
38A90A94     SYSTEM             17177              ACTIVE
38A93500      SCOTT              17227              ACTIVE
```

从上面输出知道，有两个用户通过 DRCP 连接到数据库服务器，知道其会话 ID 和进程 ID。

7.6　本章小结

本章介绍了 Oracle 网络管理的内容，首先介绍了 Oracle 网络连接原理，然后通过实例介绍如何实现动态注册、静态注册。在客户端介绍了本地命名方式和简单命名方式，二者的表现形式不同，但实质是一样的，而后介绍了数据库支持的两种连接方式：专有连接和共享连接。最后介绍了 Oracle 11g 版本专门为 Web 连接提供高效率连接服务的 DRCP 功能。

第 8 章

内存管理

本章的内存管理涉及 Oracle 数据库自身所需要的内存结构，如 SGA、PGA，这是 Oracle 数据库中最重要的内存结构。我们首先介绍 Oracle 数据库的内存架构，分别介绍这些内存结构的作用、相互关系。然后介绍内存管理以及配置。

8.1 内存架构

图 8-1 所示为 Oracle 数据库的内存架构，其中 SGA 为数据库实例的一部分，在数据库启动时会首先分配这块内存，包括数据库高速缓存、共享池、大池、流池、Java 池以及 Redo Buffer。PGA 也可以成为私有全局区，是某个用户进程所独有的。在专有连接模式下，每个用户会话都会分配一个 PGA，用户保存会话信息。当然在 PGA 与 SGA 之间是服务器进程，最终是服务器进程访问 SGA 来满足用户的数据访问。

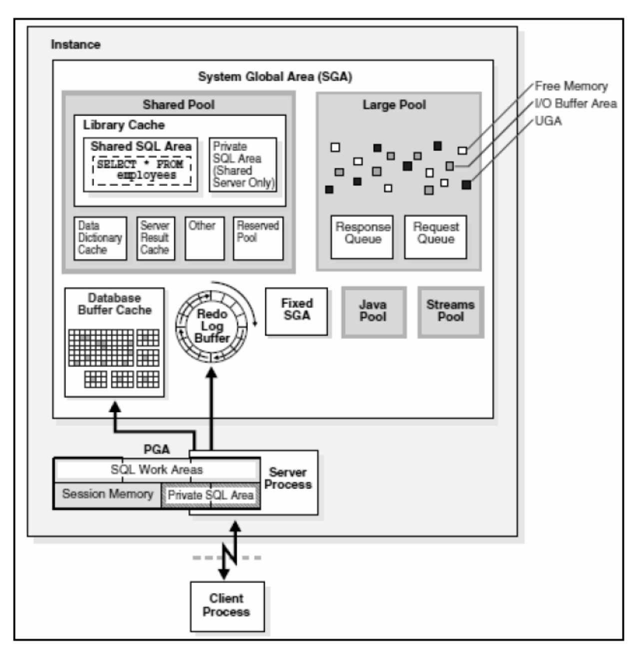


图 8-1 内存架构图

数据库实例启动时，Oracle 数据库会分配内存区并启动后台进程。内存区存储了如下的信息：

- 程序代码。
- 建立过连接的会话信息。（即使该会话当前不是活跃的 inactive。）
- 程序执行期间所需信息，如 SQL 查询的当前状态。
- 在进程间共享并且进行通信的信息如锁数据。
- 缓存数据，如数据库块、重做块。（缓存这些在磁盘上的数据。）

具体哪些内存组件完成上述任务，在详细介绍 SGA 和 PGA 时读者会有更详细的了解。

概括地讲，Oracle 数据库基本的内存结构包括：

- 系统全局区（SGA）：SGA 是一组共享的内存结构。这些组件包含数据库实例所需的数据和控制信息。它是一个共享的内存结构，所有服务器进程和后台进程共享。
- PGA 程序全局区：是非共享的内存区域，包含 Oracle 进程所独自使用的数据和控制信息。PGA 与服务器进程和后台进程共存，即 PGA 是为服务器进程和后台进程服务的内存区，为这些进程独有，不是所有进程共享的内存区，所以这部分不会出现争用。数据库初始化参数设置实例 PGA 的大小，这个大小为整个实例所拥有。
- UGA 用户全局区：该内存区域和一个用户会话相关联。
- 软件代码区：该内存区用于存储正在运行的程序代码。

8.1.1 PGA 概述

PGA 内存是专为某个操作系统进程或者线程服务的，不能被其他进程或者线程共享使用。正因为 PGA 是进程专有的内存区域，所以从来不会在 SGA 中分配。PGA 是一个内存堆，存储了专有或者共享服务器进程所需的会话相关变量。服务器进程负责在 PGA 中分配它所需要的内存结构。图 8-2 是实例 SGA 的结构图，专有服务器模式下的结构图。

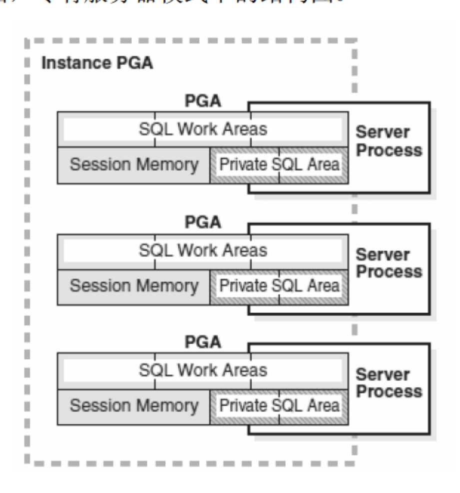


图 8-2　PGA 结构图

PGA 被细分成几个不同的区域，各自功能不同。图 8-3 是专有服务器模式下的可能的组件内

容。但是，在为一个服务器进程分配的 PGA 中并不包含所有的 PGA 组件，需要服务器进程根据 SQL 的需要来分配所需要的 PGA 组件，如图 8-3 所示。

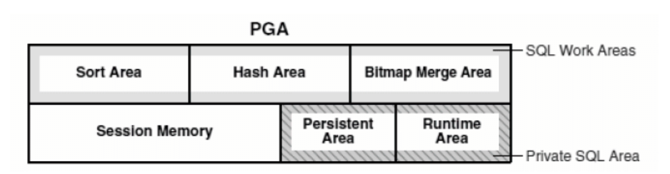


图 8-3　PGA 组件

私有 SQL 区。私有 SQL 区包含解析后的 SQL 语句和处理所需要的会话相关信息。当服务器进程执行 SQL 或 PL/SQL 代码时，该进程使用私有 SQL 区存储绑定变量的值、查询执行状态信息以及查询执行工作区。在同一个或不同会话中的多个私有 SQL 区可以指向 SGA 中的一个执行计划。此时对于每个执行计划的私有 SQL 区并不共享，各自包括不同的值或者数据。

私有 SQL 区被分成两部分：

- 运行时区：该区域包含 SQL 查询执行状态信息。如运行时区跟踪全表扫描中到目前为止返回的数据行数。在执行请求的第一步 Oracle 数据库就创建运行时区。对于 DML 语句，运行时区在 SQL 语句关闭后自动释放内存。
- 永久数据区：这部分存储绑定变量。绑定变量的值在 SQL 语句运行时赋予该语句。永久数据区只有游标关闭后方可释放。如果增加了 session_cached_cursors 的值，此时在私有 SQL 区就存储了较多的执行计划，此时与该执行计划相关的绑定变量占用空间不会释放，造成占用过的 PGA 资源，所以在 PGA 优化时要注意是否需要增加 PGA 的大小。

客户端进程负责管理私有 SQL 区。私有 SQL 区的分配和回收主要依赖于应用，客户端进程能够分配的私有 SQL 区的数量受到初始化参数 OPEN_CURSORS 的限制。尽管多数用户依赖于自动的游标管理，但是 Oracle 数据库的编程接口提供给程序员更多的游标控制。一般来讲应用在不使用游标时，要主动关闭该游标以释放永久数据区占用的内存空间，最小化用户内存需求。

SQL 工作区是一个私有内存区域，在 PGA 中分配，用于内存密集型的操作。如排序操作符使用排序区来排序行集合。类似的 hash-join 操作符使用哈希区从它的左输入构建哈希表，然而 bitmap merge 使用 bitmap merge 区来合并从多位图索引扫描获得的数据。如果操作符处理的数据总量过大，不能放入工作区，此时 Oracle 数据库将输入数据分割成更小的分片。用这种方式，数据库在内存中处理数据片，同时将其余无法处理的数据放入临时表空间待以后再处理。

数据库可以自动调整工作区的分配，如果启动了 PGA 内存自动管理。也可以手动调整工作区各个组件的大小。

一般来讲，较大的工作区可以极大提高操作符的性能，但是以较大的内存消耗为代价。如果工作区不够用，此时会出现 multiple passes，此时可能会影响 SQL 语句的执行效率。但是出现 multiple passes，也并不一定意味着性能问题。小的查询操作也可能带来 multiple passes 问题。

8.1.2　SGA 概述

SGA 是可读写的内存区。SGA 和后台进程组成了 Oracle 数据库实例，代替用户工作的服务器

进程可以读 SGA 中的数据。服务器和后台进程并不驻留在 SGA 中，而是存在于独立的内存空间中。

每一个数据库都有自己的 SGA。Oracle 数据库在实例启动时自动分配 SGA，在实例关闭时自动关闭 SGA。

```
SQL> startup
ORACLE instance started.

Total System Global Area    422670336 bytes
Fixed Size                  1336960 bytes
Variable Size               318769536 bytes
Database Buffers            96468992 bytes
Redo Buffers                6094848 bytes
Database mounted.
Database opened.
```

在数据库启动到 Database Mounted 状态之前已经启动了实例。SGA 由几个内存组件组成，称为内存池。这些内存池的空间分配以 Granules 为单位，即粒度为单位，粒度是一个连续的内存空间，粒度大小与操作系统平台有关，由整个 SGA 的大小决定。下面是最重要的几个 SGA 组件。

- Database Buffer Cache：数据库高速缓存
- Redo Log Buffer：重做日志缓存
- Shared Pool：共享池
- Large Pool：大池
- Java Pool：Java 池
- Streams Pool：流池
- Fixed SGA：固有区域

数据库高速缓存是一个共享的内存结构，这意味着发生内存争用的事件都发生共享的内存结构中。数据库高速缓存用来存储从数据文件读取的数据块，临时存储当前或者最近读取的数据块。提高 Oracle 数据库读写数据的效率。下面分析一下 Oracle 数据库如何通过设计数据库高速缓存提高读写效率：

- 优化了磁盘 I/O。数据库在该内存结构中更改数据块的内容，在重做日志缓冲区中记录这种变化，一旦用户提交，日志缓冲区中的数据就写入重做日志文件，这成为日志优先原则。但是此时的 DBWR 进程并不是立即将这个变化写入数据文件，而是等这些需要提交的数据积累到一定程度才成批写入数据文件（还有其他触发写数据文件的操作）。这种写入方式称为 LAZY 写。这样就提高了数据写入的效率。
- 将最近访问的数据块缓存在数据库高速缓存中，在一定条件下将不经常被访问的数据块写入数据文件。Oracle 遵循 LRU 数据库高速缓存算法，最近没有被使用的数据越容易被清出内存。

Oracle 数据库使用内部算法管理数据库高速缓冲区中的 Buffer 有以下几种状态。

- Unused：该缓冲区从来没有被使用过或者当前没有被使用。使用这种缓冲区对数据库而言是最容易的。
- Clean：缓冲区以前被使用过，现在包含数据块的一致性读版本。该数据块包含数据，但是 Oracle 数据库认为该缓冲区是可以被使用的，是 Clean 的。这样的缓冲区可以被 PIN 住从而重用它。

- Dirty：该缓冲区包含被修改过的数据，但是这些数据还没有被写入磁盘文件。在重用该缓冲区之前必须 Checkpoint。触发一个检验点事件。

Buffer 的访问模式有 Pinned 和 Free 两种。其中 Pinned 是正在被使用的缓冲区，这样的缓冲区不会从内存中 Aged Out。多个会话不能同时修改一个 PIN 住的缓冲区。

Oracle 数据库使用复杂的算法使得缓冲区访问高效进行。在 LRU 列表中同时存在脏缓冲区和非脏缓冲区，LRU 具有一个热端和一个冷端。冷缓冲区是最近没有被使用的缓冲区，热端是被频繁访问的缓冲区并且最近被使用过。即频繁访问的缓冲区不会被清除出 SGA。

当客户端请求数据时，Oracle 数据库以如下两种模式从数据库高速缓存获得缓冲区（其中的数据）。

- Current Mode：current mode get 也称为 db block get，指一个数据块的一次获取，这个数据块出现在当前的 Buffer Cache 中。举例：如果一个未提交的事务已经更新了某个数据块中的两行，此时就会发生一个 current mode get，它在 Buffer Cache 中未提交的行中获得数据块。即获得 Buffer Cache 中数据块的当前版本的操作。数据库在修改语句中最频繁使用 db block gets，因为这些修改语句需要更新 buffer cache 中数据块的当前版本。
- Consistent Mode：consistent read get 是数据块的一致性读版本的一次获取。这个获取可能使用 UNDO DATA。举例：如果一个未提交的事务已经更新了数据块中的两行，然后如果另一个会话发起请求查询该数据块，此时数据库使用 undo data 来构造一个 read-consistent 版本的该数据块（称为 consistent read mode）。

逻辑 I/O 也称为 Buffer I/O。指在 Buffer Cache 中对 buffers 的读写。当一个 Buffer 请求在内存中没有时，此时数据库将执行一个物理 I/O，从 Flash Cache 或者磁盘复制数据块到 buffer，然后执行一个逻辑 I/O。DBWR 进程在如下几种情况下触发写操作。

- 服务器进程找不到 Clean 的 Buffer 上，目的是将新的数据块读入数据库高速缓存。脏数据块的增加，造成空闲 Buffer 数量减少。如果空闲 Buffer 减少到内核设置的阀值，并且此时需要 Clean 的 buffer，服务器进程会给 DBWR 进程发出信号，使得其将脏数据写入磁盘。同时，Oracle 数据库使用 LRU 决定哪些脏 Buffer 需要写入磁盘。此时要维护两个队列，一个是 LRU 列表，一个是写队列。当脏 Buffer 到达 LRU 列表的冷端时，数据库将其移出 LRU 列表放入写队列。DBWR 进程将写队列中的 Buffer 写入磁盘，此时如果可能使用多快写。这样 LRU 列表的尾部就不会被脏 Buffer 阻塞，使得出现 CleanBuffer 并被新的数据快速使用。
- 在发生 Checkpoint 事件时，发生写脏 Buffer 的动作，检验点是 REDO 线程中的一个位置记录，从检验点位置开始之前的数据不需要实例恢复，这样设置合理的检验点周期获得实例恢复的时间需求。
- 表空间改为 READ-ONLY 状态或者 OFFLINE 。注意必须是该 DML 语句执行的会话进行操作才行。即当前会话修改了数据行，然后发生表空间状态变化，如 OFFLINE 然后 ONLINE 这样就会自动提交。注意这里的表空间是存储更新的表数据的表空间，非相关表空间不行。

至于每个池的作用，在第 4 章介绍过，这里不再重复。

8.1.3 UGA 概述

UGA 是会话内存，为某个用户会话服务，给会话分配内存存储会话变量，这些变量包括登录信息以及会话所需的其他信息。本质来讲 UGA 存储了用户的会话状态。图 8-4 是 UGA 组成图。

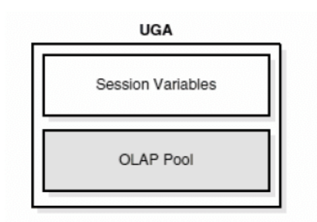


图 8-4　UGA 内存结构图

如果会话加载了 PL/SQL 包，此时该包的状态就存储在 UGA 中，这个状态是指在指定时刻包变量的值的集合。如果包的子程序改变了包变量，包的状态也会改变，此时涉及对 UGA 的修改。默认包变量在会话生命周期内是唯一并且持久存在的。

OLAP 页池也在 UGA 中存储。该池管理 OLAP 数据页。它与数据块等同。该页池随着 OLAP 会话的启动和结束而分配和释放内存。在共享服务器模式下，UGA 在 SGA 中分配，这样任何的共享服务器进程都可以访问它。如果在 PGA 中分配，由于 PGA 只为单个进程服务，不能共享。当使用专有服务器模式时，UGA 在 PGA 中分配。

8.2 内存管理

MEMORY_TARGET 设置后，SGA 和 PGA 实现自动调整，根据负载状况协调内存的使用，这个参数是动态参数，所以不需要重启数据库就可以设置，如下所示。

```
SQL>alter system set memory_target=1000M SCOPE=BOTH;
```

MEMORY_MAX_TARGET 设置 MEMORY_TARGET 的上限，毕竟操作系统的内存不仅仅为 Oracle 服务，这样就防止设置了 MEMORY_TARGET 过大的值，而使得操作系统的内存资源短缺。

如果使用 DBCA 建库，此时数据库默认启动自动内存管理。

8.2.1 配置内存组件

如果没有启动自动内存管理，比如使用 DBCA 建库时设置了 SGA 和 PGA 相关参数值，或者使用 CREATE DATABASE 手工建库时没有设置 MEMORY_TARGET 参数，此时就需要启动自动内存管理特性。

使用 AS SYSDBA 角色登录数据库，设置 MEMORY_TARGET 的值，这个值通过 SGA_TAGET+PGA_AGGREGATE_TARGET 计算。

```
memory_target = sga_target + max(pga_aggregate_target, maximum PGA
```

```
    allocated)
```

其中 sga_target 和 pga_aggregate_target 可以通过 SHOW PARAMETER TARGET 获得，而 maximum PGA Allocated 通过 select value from v$pgastat where name='maximum PGA allocated';查询获得。

一般比这个计算值还要大些，如果物理内存够的话。

MEMORY_MAX_TARGET 考虑 SGA 和 PGA 的将来预期，默认该参数与 MEMORY_TARGET 相同。它是一个静态参数，需要如下修改。

```
ALTER SYSTEM SET MEMORY_MAX_TARGET = nM SCOPE = SPFILE;
```

如果使用 PFILE 启动数据库，也可以在初始化参数文件中写入这些参数的值。

```
memory max target = nM
memory_target = mM
```

通过动态性能视图$memory_target_advice 查看对于 MEMORY_SIZE 的大小设置建议。其中内存尺寸因子 MEMORY_SIZE_FACTOR 为 1 的 MEMORY_SIZE 为当前的内存大小，也就是参数 memory_target 的大小。

例子 8-1　查看对于 MEMORY_SIZE 的大小设置建议

```
SQL> select * from v$memory target advice order by memory size;

MEMORY SIZE MEMORY SIZE FACTOR ESTD DB TIME ESTD DB TIME FACTOR   VERSION
----------- ------------------ ------------ ------------------- ----------
        202            .25           85           1.0833            0
        404            .5            80           1                 0
        808            .75           80           1                 0
        808            1             80           1                 0
       1010            1.25          80           1                 0
       1212            1.5           80           1                 0
       1414            1.75          80           1                 0
       1818            2             80           1                 0
已选择 8 行。
```

例子 8-1 中加粗部分为内存大小因子为 1 的 MEMORY_SIZE 为 808MB，完成当前负载预计的 DB 时间为 80。下面我们验证，查询 MEMORY_TARGET 的大小。

例子 8-2　查询 MEMORY_TARGET 的大小

```
SQL> show parameter memory target;

NAME                     TYPE        VALUE
------------------------------------------------ ----------------------
memory_target            big integer 808M
```

从输出知道 MEMORY_TARGET 大小为 808MB，与动态性能视图 v$memory_target_advice 中内存大小因子为 1 的 MEMORY_SIZE 大小相同。

因为我们设置了参数 MEMROY_TARGET，所以此时的 SGA_TARGET 和 PGA_TARGET 根据负载和事务情况自动调节，此时我们不需要额外调节 SGA 或者 PGA 的大小。如果设置了 SGA_TARGET 和 PGA_AGGREGATE_TARGET 的大小，则 Oracle 认为这个大小值为对应参数的最小值。因为我们启动了自动内存调节，所以此时这两个参数的值都为 0，如例子 8-3 所示。

例子 8-3　查询 PGA 和 SGA 的大小

```
SQL> show parameter sga;
NAME                                TYPE        VALUE
------------------------------------ ----------------------
lock sga                             boolean     FALSE
pre page sga                          boolean     FALSE
sga max size                          big integer    512M
sga target                           big integer    0
SQL> show parameter pga;
NAME                                TYPE        VALUE
--------------------------------------------- --------------
pga_aggregate_target                      big integer       0
```

从输出知道，PGA 和 SGA 的值为 0，说明这两个内存组件是自动调整的。在 EM 企业管理器中，可以使用图像化的内存顾问查看对于 MEMORY_SIZE 设置尺寸的建议。

8.2.2　SGA 与 PGA 的自动调整

在使用 DBCA 建库时，会出现如图 8-5 所示的对话框，选择 Custom 即可实现共享内存自动管理，需要配置 PGA 和 SGA 的大小，SGA 的内存组件是自动调整，同样 PGA 的内存组件也是自动调整。

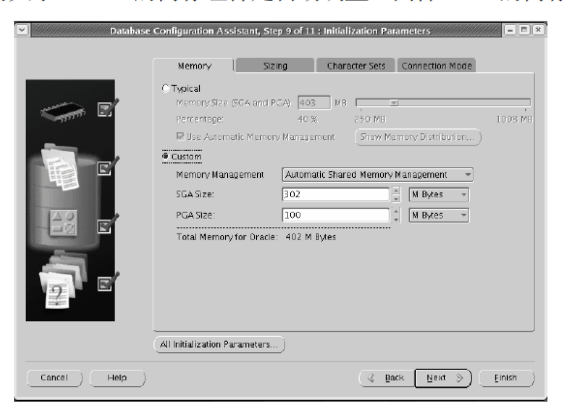


图 8-5　自动共享内存管理

对于 SGA，只要设置 TARGET 值和 MAXIMUM 值即可实现 SGA 内相关内存自动调整，当然 SGA 的其他自动调整的内存参数也可以通过手工设置。

SGA 包括共享池、大池、Java 池、数据库调整缓存、流池。内存组件名称和对应的参数名如表 8-1 所示。

表 8-1　自动调整的 SGA 内存组件与相应参数

内存组件名	参数名
共享池	SHARED_POOL_SIZE
大池	LARGE_POOL_SIZE
Java 池	JAVA_POOL_SIZE
数据库高速缓存	DB_CACHE_SIZE
流池	STREAMS_POOL_SIZE

如果使用 SGA 中内存组件完全由手工调整，可以通过设置参数 MEMORY_TARGET 和参数 SGA_TARGET 为 0 实现，此时需要手工设置共享池、Java 池、数据库高速缓存、流池和大池。这

样的方法十分复杂，不是 Oracle 推荐的方法，读者最好使用自动内存管理方式。

对于手工调整 PGA 内存组件，在较早版本中 DBA 需要调节几个*_area_size 的参数来设置 PGA 相关的内存组件，但是实际的情况不可预测，这些组件的大小设置十分困难，所以 Oracle 在 10G 以后的版本中使用了 PGA 的自动内存调整，在 Oracle11G 中这种趋势得到增强，SGA 和 PGA 也可以相关自动调整。下面是在自动 PGA 内存调整状态下 PGA 相关组件的大小信息。

例子 8-4　查询 PGA 相关组件的大小

```
SQL> show parameter area_size;

NAME                                 TYPE        VALUE
------------------------------------ ------------------------------
bitmap_merge_area_size               integer     1048578
create_bitmap_area_size              integer     8388808
hash_area_size                       integer     131072
sort_area_size                       integer     85538
workarea_size_policy                 string      AUTO
```

从这个输出知道参数 bitmap_merge_area_size、create_bitmap_area_size、hash_area_size 以及 sort_area_size 是自动调整的参数。

读者还注意到一个参数就是 workarea_size_policy，这个参数是决定自动调整 PGA 还是手工调整 PGA，如果手工调整 PGA 则必须设置参数 workarea_size_policy 为 manual。此时，就可以手动调整 4 个*_area_size 参数，但是 Oracle 还是强烈推荐使用自动内存调整，这种智能化措施十分高效，极大地减少 3DBA 的工作精力。Oracle 能做的就让 Oracle 去做。

8.2.3　配置数据库 smart flash 缓存

Smart flash cache 是 Oracle 在 11g 版本中提供的新功能，是一个新的内存组件，默认这个内存组件没有配置。也就是没有启用。它的核心作用是缓存更多的数据，提高读数据的效率，减少 CPU 的负担以及减小数据库高速缓存的压力。

使用 Smart flash cache 有几个注意事项：

- 必须是 Solaris 或者 Oracle Enterprise Linux 操作系统。
- 通过 AWR 或者 Statspck 报告提示加倍数据库高速缓存的大小。
- CPU 资源短缺。

启用 smart flash cache 功能后，从数据库高速缓存移动到 flash cache 中的数据块而言，将有部分数据块的元数据存储在数据库高速缓存中。对于单实例数据库而言，每个数据块的元数据占据大约 100B 的空间。而对于 RAC 环境则大约是 200B，所以在设置 smart flash cache 时需要考虑数据库高速缓存的额外空间需求。

如果是手工调整内存方式，则需要增加数据库高速缓存的大小，其值为进入 smart flash cache 的数据块数量乘以 100。如果采用自动内存调整则需要调整 MEMORY_TARGET，大小参考手工调整内存的大小。如果使用自动共享内存调整，即 SGA 组件自动调整，此时需要增加 SGA_TARGET 的值。

设置 smart flash cache 时需要设置两个参数，其中一个参数需要指定存储 smart flash cache 数据

块的磁盘目录以及名称，要求必须在 Flash 盘上存储 smart cache 数据，否则会影响性能。涉及的两个参数是 db_flash_cache_file 和 db_flash_cache_size，如例子 8-5 所示。

例子 8-5　查询 SmartFlash Cache 涉及的两个参数

```
SQL> show parameter flash cache;

NAME TYPE VALUE
------------------------------------------------------------------------
db flash cache file                  string
db_flash_cache_size                  big integer 0
```

修改这两个参数以启动 smart flash cache 功能。

例子 8-6　设置 db_flash_cache_file 和 db_flash_cache_size 参数

```
SQL> alter system set db flash cache file='/u01/flash cache' scope=spfile;

System altered.


SQL> alter system set db flash cache size=1g scope=spfile;

System altered.
```

这两个参数都不是动态参数，修改后需要重启数据库才能生效。我们重启数据库后验证修改结果。

例子 8-7　重启数据库使参数修改生效

```
SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORA-00439: feature not enabled: Server Flash Cache
```

我们看到此时的提示为该特性没有启动。其实，这个也在预想之中，因为我们的操作系统不是启动该特性要求的系统，显然不会得到支持。如果在 Solaris 或者 Oracle Enterprise Linux 系统上将得到支持，但是设置方法是一样的。

8.3　本章小结

本章我们学习了 Oracle 数据库的内存组件，主要涉及 SGA 和 PGA 内存组件的结构、各个子组件的功能以及注意事项，通过内存管理部分知道如何设置内存的自动管理和手动管理，以完成初始的数据库内存配置任务。

第 9 章

用户管理和资源文件

Oracle 通过设置用户来访问数据库，对用户赋予不同的资源使得用户具有操作数据库的不同权限。在数据库被使用前，必须由数据库管理员创建用户和用户登录密码，随后可以由 DBA 或其他具有 DBA 权限的用户授予新创建的用户访问各种资源的权利。

9.1 创建用户

在 Oracle 中必须使用用户登录数据库，通过设置密码完成用户身份认证，一旦登录数据库，该用户就可以访问它拥有的数据库对象，最明显的例子是访问其中的表对象。下面我们先给出一个例子说明如何创建用户，使得读者对用户创建有直观认识，然后再通过创建用户的语法详细介绍一些参数设置。

9.1.1 初试创建新用户

要创建数据库必须使用 DBA 权限的用户，本例中使用 SYS 用户登录数据库，密码为 Oracle1234，这个密码是需要复杂度的。读者的密码或许有所不同，在安装 Oracle 11g 数据库过程中会提示解锁的用户设置密码。

例子 9-1 使用 SYS 用户登录数据库

```
[oracle@myoracle ~]$ sqlplus sys/Oracle1234 as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Tue Jan 1 22:59:26 2013

Copyright (c) 1992, 2013, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

例子 9-2 创建数据库用户

```
SQL> create user jane
  2  identified by american
```

```
  3  default tablespace users
  4  temporary tablespace temp
  5  quota 10m on users
  6  password expire;
```

在例子 9-2 中，我们创建了用户 JANE，下面我们具体分析每行的含义。

- create user jane：创建用户 JANE，其中 create user 为创建用户指令。
- identified by american：设置用户密码为 american，其中 identified by 为创建用户指令。
- default tablespace users：设置默认表空间为 USERS 表空间，该表空间存储用户数据。
- temporary tablespace temp：创建临时表空间 TEMP，该表空间用户诸如排序等操作的数据空间。
- quota 10m on users：设置该用户对于表空间 USERS 的配额为 10M。
- password expire：说明用户 JANE 登录数据库时，密码立即失效，Oracle 会提示重新输入密码，如下所示。

```
SQL> connect jane/american@orcl
ERROR:
ORA-29001: the password has expired


Changing password for jane
New password:
Retype new password:
ERROR:
ORA-01045: user JANE lacks CREATE SESSION privilege; logon denied


Password changed
```

此时，由于在创建用户 JANE 时，没有赋予该用户 CREATE SESSION 权利，所以虽然更改了用户密码，还是无法建立与数据库的会话连接。需要向用户授权，否则该用户就是“花瓶”（中看不中用），在赋予该用户权限后，该用户就可以使用新的用户密码登录数据库了，如例子 9-3 所示。

例子 9-3　为用户 JANE 赋予 CREATE SESSION 权限

```
SQL> connect sys/Oracle1234 as sysdba
Connected.
SQL> grant create session ,resource to jane;

Grant succeeded.
```

然后就可以用新密码登录数据库了，如下所示。

```
SQL> connect jane/oracle@orcl
Connected.
```

在成功创建了数据库后如何查看创建的用户诸如表空间等信息呢，答案是使用数据字典 DBA_USERS。此时，需要以 DBA 角色的用户登录数据库，如例子 9-4 所示。

例子 9-4　使用数据字典 DBA_USERS 查看用户 JANE 的信息

```
    SQL> col username for a10
    SQL> col default_tablespace for a10
    SQL> col temporary_tablespace for a15
    SQL> col password for a20
    SQL> select username,password,expiry_date,default_tablespace,temporary
tablespace,created
      2  from dba_users
      3* where username = 'JANE'

    USERNAME   PASSWORD   EXPIRY_DATE   DEFAULT_TA TEMPORARY_TABLE CREATED
    ------------------------------------------------------------------------
    JANE                  30-JUN-13        USERS      TEMP             01-JAN-13
```

从例子 9-4 的输出可以看出用户的默认表空间为 USERS，而临时表空间为 TEMP，该用户的创建时间为 01-JAN-13。而密码是加密的，这也是 Oracle 认为安全第一的缘故，即使具有 DBA 权限的用户也无法看到该用户的密码，虽然 DBA 用户可以创建或删除用户。

在创建用户时，我们使用了 QUOTA 参数，设置该用户只能使用表空间 USERS 的 10M 空间，使用数据字典 DBA_TS_QUOTAS 可以查看用户 JANE 在表空间上的配额信息。

例子 9-5　查看用户 JANE 的表空间配额信息

```
    SQL> select tablespace_name,username, max_bytes
      2  from dba_ts_quotas
      3  where username ='JANE';

    TABLESPACE_NAME     USERNAME     MAX_BYTES
    ----------------------------------------------------- ------------------
    USERS               JANE          10495760
```

9.1.2　创建用户语法及参数含义

在 9.1.1 节我们通过例子体验了如何创建一个新用户，其实读者只需要修改个别参数如表空间的名字（可能需要用户事先创建表空间）就可以直接应用来创建自己的用户。下面是创建用户的详细语法。

例子 9-6　创建用户的语法格式

```
    CREATE USER user
    IDENTIFIED {BY password | EXTERNALLY}
    [DEFAULT TABLESPACE tablespace]
    [TEMPORARY TABLESPACE tablespace]
    [QUOTA {integer [K | M]| UNLIMITED} ON tablespace
    [QUOTA {integer [K | M]| UNLIMITED} ON tablespace
    ]……]
    [PASSWORD EXPIRE]
    [ACCOUNT { LOCK | UNLOCK }]
    [PROFILE { profile | DEFAULT }]
```

下面介绍其中的几个参数：

- CREATE USER user：创建用户 user。
- IDENTIFIED{BY password | EXTERNALLY}：设置用户密码，EXTERNALLY 说明该用户由操作系统授权。该参数在创建用户时是不能省略的。
- DEFAULT TABLESPACE tablespace：设置用户的默认表空间。
- TEMPORARY TABLESPACE tablespace：设置用户的临时表空间。
- QUOTA {integer[K|M]|UNLIMITED} ON tablespace：设置该用户对于表空间的配额，即表空间的多大空间给该用户使用，参数 UNLIMITED 说明没有限制，K|M 是配额单位。
- PASSWORD EXPIRE：设置用户密码在用户第一次使用时作废，需要重新设置该用户密码。
- ACCOUNT{LOCK | UNLOCK}：选择是否锁定该用户，LOCK 锁定用户，而 UNLOCK 不锁定用户，该参数的默认值是 UNLOCK。
- PROFILE {profile |DEFAULT}：使用指定的概要文件，profile 为概要文件名。如果不指定概要文件，则使用 DEFAULT 的默认概要文件，默认的概要文件对所有限制的初始值都没有限制。

从以上创建用户的语法可以看出，在创建新用户前，必须做些准备工作，整个准备工作和创建过程如下所示。

- 确认存储用户对象的表空间。
- 确定在每个表空间上的空间配额。
- 分配一个默认表空间和一个临时表空间。
- 开始创建用户。
- 向用户授权和角色。如使得用户具有建立会话的权利，辅以用户 DBA 角色权限等。

9.1.3 改变用户参数

在成功创建用户后，如果对用户参数如默认表空间等不满意，可以改变用户参数，如修改用户 JANE 的默认表空间或者修改当前默认表空间的配额。

例子 9-7 修改用户 JANE 的默认表空间配额

```
SQL> connect system/oracle@orcl
Connected.
SQL> alter user jane
  2  quota 20m on users;

User altered.
```

在修改成功后，我们使用数据字典 DBA_TS_QUOTAS 来验证修改结果。

例子 9-8 验证用户 JANE 的表空间修改结果

```
SQL> select tablespace name,username,max bytes
  2  from dba ts quotas
  3  where username ='JANE';
```

```
TABLESPACE NAME          USERNAME          MAX BYTES
-------------------------------------------------------------------------
USERS                      JANE             20971520
```

从输出结果可以看出用户 JANE 在表空间 USERS 上的配额被修改为 20M。说明例子 9-7 的修改成功。

在生产数据库中，会出现用户默认表空间不足的情况，在用户创建数据库对象如表、索引时，如果没有指定存储表空间就存放在创建用户时的默认表空间中，数据表空间的不足会造成数据库挂起，所以需要修改用户在默认临时表空间的 QUOTA 参数，而如何增加一个默认表空间呢，下面给出一个示例（用户事先创建表空间 NEWTBS）。

例子 9-9　修改用户 JANE 的默认表空间

```
SQL> alter user jane
  2  default tablespace newtbs
  3  quota unlimited on system;

用户已更改。
```

例子 9-9 中，我们增加了用户 JANE 的一个默认表空间为 NEWTBS，在该表空间的配额为 UNLIMITED（没有限制）。我们再通过数据字典 DBA_TS_QUOTAS 来查看修改结果。

例子 9-10　查看用户 JANE 的默认表空间修改信息

```
SQL> select username,tablespace name,max bytes
  2  from dba ts quotas
  3  where username ='JANE';

USERNAME         TABLESPACE_NAME              MAX_BYTES
-------------------------------------------------------------------------
JANE                 NEWTBS                        -1
JANE                 USERS                     20971520
```

从上述输出可以看出用户 JANE 在表空间 NEWTBS 上的配额为-1，说明没有限制，而此时用户在表空间 USERS 上的配额依然存在。而如果不希望用户使用表空间 USERS 的空间，即回收用户在 USERS 表空间的使用权，又如何处理呢，如例子 9-11 所示。

例子 9-11　回收用户 JANE 在表空间 USERS 的使用权

```
SQL> alter user jane
  2  quota 0 on users;

用户已更改。
```

此时，我们使用设置用户在表空间 USERS 上的配额为 0 来回收对其使用权。然后我们再通过数据字典 DBA_TS_QUOTAS 来查看该用户的表空间配额信息，如例子 9-12 所示。

例子 9-12　验证是否回收用户 JANE 的表空间 USERS 的使用权

```
SQL>  select username,tablespace_name,max_bytes
  2   from dba_ts_quotas
  3   where username ='JANE';
```

```
USERNAME                    TABLESPACE_NAME              MAX_BYTES
--------------------------------------------------------------------------
JANE                         NEWTBS                              -1
```

从输出可以看出用户 JANE 没有使用表空间 USERS，只有表空间 NEWTBS，而且该表空间的使用空间不受限制（当然不能超过表空间中所有数据文件大小的总和，即使数据文件的大小可以自动扩展也不能超过磁盘空间的限制☺），如果在回收 USERS 表空间的使用权之前，已经在该表空间上使用了 5M 空间，则不能再给用户 JANE 分配使用空间了。

9.2　删除用户

删除用户的语法格式如下所示。

```
DROP USER user_name [CASCADE]
```

如果使用 CASCADE 参数说明要删除掉和用户相关的所有数据库对象，如触发器、外键索引、过程等。我们删除用户 JANE，如例子 9-13 所示。

例子 9-13　删除用户 JANE

```
SQL> drop user jane;

User dropped.
```

下面，我们验证是否成功删除用户 JANE，如例子 9-14 所示。

例子 9-14　验证用户 JANE 是否存在

```
SQL> select username,created,default_tablespace
  2  from dba_users
  3  where username ='JANE';

no rows selected
```

输出结果是“未选定行”，说明用户 JANE 不存在，在删除用户时，如果该用户已经连接到数据库服务器，则无法删除。可以断开该用户的连接在删除用户。

我们也可以使用数据字典 DBA_USERS 来查看当前系统上的用户名，如例子 9-15 所示。

例子 9-15　查看当前系统上的所有用户信息

```
SQL> select username,account_status,created
  2  from dba_users;

USERNAME                     ACCOUNT_STATUS                 CREATED
----------------------------------------------------- ------------
LAURENCE                   OPEN                             04 -SEP -12
SCOTT                      OPEN                             25-AUG -12
CAT                        OPEN                             9-AUG -12
HR                         OPEN                             12-JUL -12
RMAN_BACKUP                OPEN                             03-SEP -12
```

```
TSMSYS                 EXPIRED & LOCKED                  30-AUG -12
BI                     EXPIRED & LOCKED                  12-AUG -12
PM                     EXPIRED & LOCKED                  12- AUG -12
…………
OUTLN                  EXPIRED & LOCKED                  30-AUG -12

已选择 31 行。
```

在以上输出中 ACCOUNT_STATUS 说明用户的状态，其中值 OPEN 说明该用户可用，而 EXPIRED 说明该用户过期，LOCKED 说明该用户锁定。那么如何解锁这些锁定的用户呢，如例子 9-16 所示。

例子 9-16　解锁用户

```
SQL> ALTER USER outln IDENTIFIED BY outln ACCOUNT UNLOCK;

用户已更改。
```

解锁过程中，我们需要先使用 IDENTIFIED BY 修改用户的密码，这也是 Oracle 安全理念的体现，无论何时安全第一。解锁了的用户就可以正常登录数据库了，如例子 9-17 所示。

例子 9-17　使用解锁的用户登录数据库

```
SQL> connect outln/outln@orcl
已连接。
SQL> show user
USER 为 "OUTLN"
```

9.3　用户和数据库模式

在创建了用户后，需要赋予该用户权限使得它可以创建数据库对象如表、索引、触发器等，而此时会涉及模式的概念。模式与用户对应，当一个用户创建成功时，也对应地创建了一个模式。而且二者是一对一的关系，名字相同。

模式是命名的数据库对象的集合，这些数据库对象包括表、视图、索引等。用户拥有数据库的模式，而且用户名和模式经常互换使用。下面给出模式对象：

- 表
- 触发器
- 约束
- 索引
- 视图
- 序列号
- 存储过程
- 同义词
- 用户定义的数据类型
- 函数
- 软件包

我们可以使用 SCOTT 用户登录数据库，然后使用数据字典 USER_OBJECTS 来查看当前用户所拥有的数据库模式对象，如例子 9-18 所示。

例子 9-18　查看用户 SCOTT 所拥有的模式对象

```
SQL>  select distinct(object type) from user objects;

OBJECT_TYPE
-------------------
INDEX
TABLE
```

从输出可以看出 SCOTT 用户或称 SCOTT 模式拥有 2 个模式对象，分别是表（TABLE）和索引（INDEX）。

9.4　用户管理中的重要文件——概要文件

在创建用户后就需要给该用户各种系统资源，如 CPU、并行会话数、空闲时间限制等资源限制，同时需要对口令做出更详细的管理方案，如尝试登录指定的次数后账户被锁定、口令过期之后的处理等，如果对每个用户都进行资源限制或口令管理，要输入大量的指令，比如每个用户输入 10 条资源限制或口令限制指令，对 10 个用户就输入 100 条指令，显然这样的效率很低，尤其是对用户的资源限制和口令限制都相同时，只是重复的输入指令，Oralce 提供了概要文件来管理用户，可以避免上述问题。

9.4.1　什么是概要文件

概要文件就是一组指令的集合，这些指令限制了用户资源的使用或口令的管理，在创建用户时，有一个 PROFILE 参数就是用来指定概要文件的，一旦概要文件创建就可以将概要文件通过 ALTER USER 指令赋予用户或者在 CREATE USER 时指定概要文件。

通过将概要文件赋予用户可以极大较少 DBA 的工作量。如果没有指定概要文件，则会自动使用一个默认概要文件。

9.4.2　使用资源管理和口令管理的概要文件步骤

使用概要文件可以实现用户的资源管理和口令管理。使用步骤如下所示。

- 使用 CREATE PROFILE 指令创建一个概要文件。
- 使用 ALTER USER（已有用户）或 CREATE USER（新用户）将概要文件赋予用户。
- 对于资源管理而言，启动资源限制，修改动态参数 RESOURCE_LIMIT 为 TRUE，此时既可以通过修改参数文件也可以使用 ALTER SYSTEM 来修改。

9.4.3　使用概要文件管理会话资源

当用户连接到数据库时，就与数据库服务器建立了会话连接，此时用户会消耗数据库服务器的资源，所以我们创建一个会话级的数据库资源限制的概要文件来限制用户对资源的使用。我们先给出创建资源管理的概要文件的语法格式，如下所示。

```
CREATE PROFILE  profile_name LIMIT
[SESSIONS PER USER n]
[CPU PER SESSION n]
[CPU PER CALL n]
[CONNECT_TIME n]
[IDLE_TIME n]
[LOGICAL READS PER SESSION n]
[LOGICAL_READS_PER_CALL n]
```

其中 n 为最大值。下面我们逐行介绍每个资源限制的含义。

- SESSIONS_PER_USER n：表示每个用户的最大会话数。
- CPU_PER_SESSION n：每个会话占用的 CPU 时间，单位是 0.01 秒。
- CPU_PER_CALL n：每个调用占用的 CPU 时间，单位是 0.01 秒。
- CONNECT_TIME n：每个连接支持连接的时间。
- IDLE_TIME n：每个会话的空闲时间。
- LOGICAL_READS_PER_SESSION n：每个会话的物理和逻辑读数据块数。

下面我们创建一个资源限制文件。

例子 9-19　创建资源限制概要文件

```
SQL> create profile scott prof limit
  2  sessions per user 10
  3  cpu_per_session 10000
  4  idle_time 40
  5  connect time 120;

配置文件已创建
```

该资源限制文件创建了一个名为 SCOTT_PROF 的概要文件，加在该文件上的限制是 sessions_per_user 每个用户的并行会话数为 10，cpu_per_session 每个会话的 CPU 时间为 1000 秒。idle_time 连接空闲时间为 40 分，connect_time 保持连接时间为 120 分。

我们通过数据字典 DBA_PROFILES 来查看刚刚创建的概要文件 SCOTT_PROF。

例子 9-20　查看概要文件 SCOTT_PROF

```
SQL> col profile for a20
SQL> col resource name for a25
SQL> col limit for a20
SQL> select *
  2  from dba profiles
  3  where profile ='SCOTT PROF'
  4  order by limit;
```

```
PROFILE              RESOURCE_NAME                 RESOURCE    LIMIT
---------------------------------------- --------- ---------------------
SCOTT PROF           SESSIONS PER USER             KERNEL      10
SCOTT PROF           CPU PER SESSION               KERNEL      10000
SCOTT_PROF           CONNECT_TIME                  KERNEL      120
SCOTT_PROF           IDLE_TIME                     KERNEL      40
SCOTT PROF           LOGICAL READS PER SESSION     KERNEL      DEFAULT
SCOTT PROF           LOGICAL READS PER CALL        KERNEL      DEFAULT
SCOTT PROF           PASSWORD GRACE TIME           PASSWORD    DEFAULT
SCOTT_PROF           PRIVATE_SGA                   KERNEL      DEFAULT
SCOTT_PROF           COMPOSITE_LIMIT               KERNEL      DEFAULT
SCOTT PROF           PASSWORD LIFE TIME            PASSWORD    DEFAULT
SCOTT PROF           PASSWORD REUSE TIME           PASSWORD    DEFAULT

PROFILE              RESOURCE_NAME            RESOURCE LIMIT
---------------------------------------- --------- ---------------------
SCOTT PROF           PASSWORD REUSE MAX            PASSWORD    DEFAULT
SCOTT PROF           PASSWORD VERIFY FUNCTION      PASSWORD    DEFAULT
SCOTT_PROF           PASSWORD_LOCK_TIME            PASSWORD    DEFAULT
SCOTT_PROF           CPU_PER_CALL                  KERNEL      DEFAULT
SCOTT PROF           FAILED LOGIN ATTEMPTS         PASSWORD    DEFAULT

已选择 16 行。
```

从输出可以看出概要文件 SCOTT_PROF 的所有资源参数，其中资源参数 SESSION_PER_USER、CPU_PER_SESSION、CONNECT_TIME、IDLE_TIME 为创建概要文件时指定的值，而其他资源参数都采用默认值。其中 RESOURCE 列的值中 KERNEL 行表示一个资源参数，而 PASSWORD 表示一个安全限制，接下来我们介绍如何创建口令管理的概要文件。

9.4.4 口令管理参数以及含义

创建口令管理的概要文件与创建资源限制的概要文件一样，都是使用 CREATE USER 或者 ALTER USER 指令将概要文件赋予用户，口令文件一旦赋予用户立即生效，不需要开启设置。下面介绍完成口令管理的参数以及含义。首先，查看概要文件 SCOTT_PROF 中的口令参数，重新使用数据字典 DBA_PROFILES 查看概要文件 SCOTT_PROF，如例子 9-21 所示。

例子 9-21　查看 SCOTT_PROF 概要文件的口令管理参数

```
SQL> col resource_name for a30
SQL> select *
  2  from dba profiles
  3  where profile='SCOTT_PROF'
  4* and resource_type='PASSWORD'

PROFILE        RESOURCE NAME      RESOURCE TYPE            LIMIT
--------------------------------------------------------------------------
SCOTT_PROF     FAILED_LOGIN_ATTEMPTS        PASSWORD       DEFAULT
SCOTT_PROF     PASSWORD_LIFE_TIME           PASSWORD       DEFAULT
SCOTT_PROF     PASSWORD_REUSE_TIME          PASSWORD       DEFAULT
```

```
SCOTT_PROF   PASSWORD_REUSE_MAX          PASSWORD        DEFAULT
SCOTT_PROF   PASSWORD_VERIFY_FUNCTION    PASSWORD        DEFAULT
SCOTT_PROF   PASSWORD_LOCK_TIME          PASSWORD        DEFAULT
SCOTT_PROF   PASSWORD_GRACE_TIME         PASSWORD        DEFAULT

已选择 7 行。
```

从输出发现有 7 个参数实现用户的口令管理，如下所示。

- FAILED_LOGIN_ATTEMPTS：尝试失败登录的次数，如果用户登录数据库时登录失败次数超过该参数的值则锁定该用户。
- PASSWORD_LIFE_TIME：口令有效的时限，超过该参数指定的天数则口令失效。
- PASSWORD_REUSE_TIME：口令在能够重用之前的天数。
- PASSWORD_REUSE_MAX：口令能够重用之前的最大变化数。
- PASSWORD_VERIFY_FUNCTION：在为一个新用户赋予口令之前要验证口令的复杂性是否满足要求的一个函数，该函数使用 PL/SQL 语言编写，名字为 verify_function。
- PASSWORD_LOCK_TIME：当用户登录失败后，用户被锁定的天数。
- PASSWORD_GRACE_TIME：口令过期之后还可以继续使用的天数。

口令的最小长度要求 4 个字符。

口令不能与用户名相同。

口令应至少包含一个字符、一个数字和一个特殊字符。数字包括'0123456799'等。

字符包括'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'等，而特殊字符包括'!"#$%&()``*+,-/:;<=>?_'等。

新口令至少有 3 个字母与旧口令不同。

要使用 Oracle 提供的口令验证函数，需要先运行一个名为 utlpwdmg.sql 的脚本文件，执行脚本文件创建口令复杂性验证函数时，需要使用 SYS 用户登录数据库且作为 DBA 用户，该文件在$ORACLE_HOME\RDBMS\ADMIN 目录下（根据安装的磁盘略有不同）。

如下是执行该脚本文件的过程。

例子 9-22　执行创建口令复杂性验证函数的过程

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL> @ F:\app\oracle\product\11.2.0\dbhome_1\RDBMS\ADMIN\utlpwdmg.sql

函数已创建。


配置文件已更改
```

从输出可以看出，函数已经创建，且配置文件已经更改，这里创建了函数 VERIFY_FUNCTION。我们查看该函数是否存在，如例子 9-23 所示。

例子 9-23　验证口令验证函数 VERIFY_FUNCTION 是否创建

```
SQL> col owner for a10
SQL> col object_name for a20
```

```
SQL> select owner ,object name,object type,created
  2  from dba_objects
  3  where object_type ='FUNCTION'
  4* and object name ='VERIFY FUNCTION'

OWNER      OBJECT_NAME         OBJECT_TYPE         CREATED
------------------------ ------------------------------- --------------
SYS        VERIFY_FUNCTION     FUNCTION            05-9月 -12
```

显然，函数 VERIFY_FUNCTION 存在说明创建成功。那么“配置文件已经更改”是什么意思呢？也就是说在例子 9-22 中不但创建了函数，而且还更改了默认的概要文件，此时查看脚本文件 utlpwdmg.sql 就一目了然了，如下代码是脚本中最后部分。

```
-- This script alters the default parameters for Password Management
-- This means that all the users on the system have Password Management
-- enabled and set to the following values unless another profile is
-- created with parameter values set to different value or UNLIMITED
-- is created and assigned to the user.

ALTER PROFILE DEFAULT LIMIT
PASSWORD LIFE TIME 60
PASSWORD_GRACE_TIME 10
PASSWORD_REUSE_TIME 1900
PASSWORD REUSE MAX UNLIMITED
FAILED LOGIN ATTEMPTS 3
PASSWORD_LOCK_TIME 1/1440
PASSWORD_VERIFY_FUNCTION verify_function;
```

说明部分已经说得很清楚了，这部分脚本代码改变了口令管理的默认概要文件，这意味着整个数据库系统的用户都使用在 ALTER PROFILE DEFAULT LIMIT 中设置的口令限制，除非用户创建了另一个口令管理的概要文件，或修改了概要文件参数值。

此时我们更改用户 SCOTT 的用户密码为 oracle，看是否成功修改。

例子 9-24　修改用户 SCOTT 的用户密码为 oracle

```
SQL> conn system/oracle as sysdba
已连接。
SQL> alter user scott
  2  identified by oracle
  3  ;
alter user scott
*
第 1 行出现错误:
ORA-29003: 指定口令的口令验证失败
ORA-20002: Password too simple
```

显然，修改失败，因为密码 oracle 不符合函数 VERIFY_FUNCTION 中定义的规则之一。此时也说明口令管理的概要文件即时生效。

使用 Oracle 的口令函数一般可以满足用户要求，如果用户需要也可以自己创建口令函数，这里不再详细介绍，读者可以参考其他书籍。

9.4.5　创建口令管理的概要文件

在介绍了口令管理的参数以及含义后，我们就可以根据业务需要创建口令管理概要文件。如同创建资源限制的概要文件一样，语法格式如下所示。

```
CREATE PROFILE profile name LIMIT
[parameter1 para value1]
[parameter2 para_value2]
…
```

其中的 parameter1、parameter2……就是在 9.4.4 节中介绍的 7 个参数。

例子 9-25　创建口令管理的概要文件

```
SQL> create profile password_prof limit
  2  failed_login_attempts 5
  3  password life time 90
  4  password reuse time 30
  5  password_lock_time 15
  6  password_grace_time 3;

配置文件已创建
```

上面创建了概要文件 password_prof，各个参数的含义如下所示。

- failed_login_attempts 5：尝试登录的失败次数为 5 次，5 次之后该用户将被锁定。
- password_life_time 90：该密码在 90 天内有效。
- password_reuse_time 30：该口令失效后 30 天后才可以使用。
- password_lock_time 15：在尝试登录指定的次数后，该用户被锁定 15 天。
- password_grace_time 3：在口令过期后，4 天内可以使用旧口令（过期的口令）登录数据库。

现在读者已经明白了例子 9-25 中创建的密码概要文件的作用了，我们通过数据字典 DBA_PROFILES 查看密码概要文件 PASSWORD_PROF 的口令参数设置。

例子 9-26　查看密码概要文件 PASSWORD_PROF 的口令参数

```
SQL> col resource type for a10
SQL> col limit for a15
SQL>select *
  2   from dba_profiles
  3   where profile ='PASSWORD PROF'
  4*  and resource type ='PASSWORD'

PROFILE             RESOURCE_NAME             RESOURCE_T LIMIT
--------------------------------------------------------------------------------
PASSWORD PROF     FAILED LOGIN ATTEMPTS     PASSWORD   5
PASSWORD PROF     PASSWORD LIFE TIME         PASSWORD   90
PASSWORD_PROF     PASSWORD_REUSE_TIME       PASSWORD   30
PASSWORD_PROF     PASSWORD_REUSE_MAX       PASSWORD  DEFAULT
PASSWORD PROF     PASSWORD VERIFY FUNCTION PASSWORD   DEFAULT
PASSWORD_PROF     PASSWORD_LOCK_TIME        PASSWORD   15
```

```
PASSWORD_PROF     PASSWORD_GRACE_TIME      PASSWORD   3

已选择 7 行。
```

从例子 9-26 的输出可以看出在创建密码概要文件时，没有明确给出数值的都采用默认值，这些参数 LIMIT 列的值为 DEFAULT。

9.5　修改和删除概要文件

Oracle 允许使用 ALTER PROFILE 指令来修改概要文件中的参数，我们修改概要文件 PASSWORD_PROF 的口令管理参数。

例子 9-27　修改口令管理概要文件的参数

```
SQL> alter profile password_prof limit
  2  failed_login_attempts 3
  3  password_life_time 60
  4  password_grace_time 7;

配置文件已更改
```

输出显示成功修改口令管理的配置文件，下面我们使用数据字典 DBA_PROFILES 来验证修改结果。

例子 9-28　查看修改后的口令管理概要文件 PASSWORD_PROF 的参数

```
SQL>  select *
  2   from dba_profiles
  3   where profile ='PASSWORD_PROF'
  4  and resource_type ='PASSWORD';

PROFILE          RESOURCE_NAME               RESOURCE_T LIMIT
---------------------------------------- ------------------- --------------
PASSWORD_PROF    FAILED_LOGIN_ATTEMPTS       PASSWORD    3
PASSWORD_PROF    PASSWORD_LIFE_TIME          PASSWORD    60
PASSWORD_PROF    PASSWORD_REUSE_TIME         PASSWORD    30
PASSWORD_PROF    PASSWORD_REUSE_MAX          PASSWORD    DEFAULT
PASSWORD_PROF    PASSWORD_VERIFY_FUNCTION    PASSWORD    DEFAULT
PASSWORD_PROF    PASSWORD_LOCK_TIME          PASSWORD   15
PASSWORD_PROF    PASSWORD_GRACE_TIME         PASSWORD   7

已选择 7 行。
```

上述输出中参数被修改成功。

如果不需要某个概要文件，可以使用指令 DROP PROFILE 删除，如果要删除的概要文件已经赋予了用户则需要使用 CASCADE 参数。如例子 9-29 所示删除概要文件 PASSWORD_PROF。

例子 9-29　删除概要文件 PASSWORD_PROF

```
SQL> drop profile password_prof;
配置文件已删除。
```

下面验证是否成功删除口令管理的概要文件 PASSWORD_PROF。如例子 9-30 所示。

例子 9-30　验证是否删除概要文件 PASSWORD_PROF

```
SQL> select *
  2  from dba_profiles
  3  where profile ='PASSWORD_PROF';

未选定行
```

“未选定行”说明成功删除了概要文件 PASSWORD_PROF，因为在数据字典 DBA_PROFILES 中没有该文件记录。

9.6　本章小结

用户管理是 Oracle 实现安全管理的重要部分，通过创建一个新用户，并赋予一定的用户权限以访问数据库资源，通过设置用户密码提供用户登录数据库的安全验证，Oracle 通过设置密码管理的概要文件来方便管理用户密码，通过资源管理的概要文件来赋予用户使用不同的数据库资源，如每个会话使用的 CPU 时间、每个会话的连接时间等，这些概要文件可以赋予不同的用户。根据业务需求，用户可以使用 ALTER PROFILE 指令更改概要文件的参数，或使用 DROP PROFILE 删除不需要的概要文件。

第 10 章

控制文件和数据库启动

在数据库的启动过程中需要打开控制文件，控制文件中保存了 Oracle 系统需要的其他文件的存储目录和物理数据库相关的状态信息。Oracle 系统利用控制文件打开数据库文件、重做日志文件等从而最终打开数据库。本章着重讲解控制文件在数据库启动过程中的作用，以及如何维护控制文件、实现控制文件的多重控制、添加控制文件以及备份和恢复控制文件，本章的内容对于 Oracle 9i、10g 以及 11g 的版本都适用，或许文件目录有所差别，但是维护和管理控制文件的方法是一样的。

10.1 控制文件和数据库启动概述

对于 DBA 来讲，Oracle 数据库控制文件是相当重要的文件，它是一个非常小的二进制文件，其中记录了数据库的状态信息，如重做日志文件与数据文件的名字和位置、归档重做日志的历史等，它的大小不会超过 64MB，但是归档日志的历史记录会让该文件逐渐变大。

控制文件在数据库启动的 MOUNT 阶段被读取，一个控制文件只能与一个数据库相关联，即控制文件和数据库是一对一的关系，因为控制文件的重要性，所以需要将控制文件放在不同磁盘上，以防止控制文件的失效造成数据库无法启动，控制文件的大小在 CREATE DATABASE 语句中被初始化。

数据库启动与控制文件的关系如图 10-1 所示。

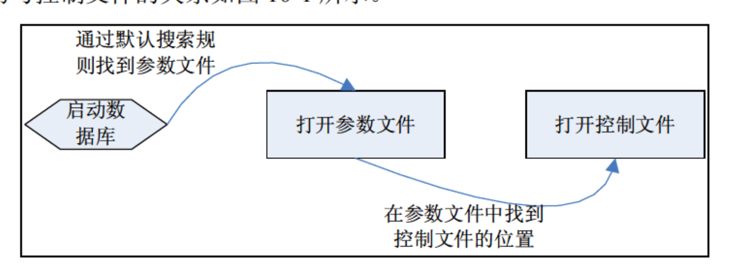


图 10-1 数据库启动和控制文件的关系

图 10-1 说明了数据库启动和控制文件的关系，也说明了数据库启动时读取文件的顺序，在数据库启动时，会首先使用默认的规则找到并打开参数文件，在参数文件中保存了控制文件的位置信息（当然还有内存分配等的信息），通过参数文件 Oracle 可以找到控制文件的位置，打开控制文件，然后会通过控制文件中记录的各种数据库文件的位置打开数据库，从而启动数据库到可用状态。

当成功启动数据库后，在数据库的运行过程中，数据库服务器可以不断地修改控制文件中的内容，所以在数据库被打开的阶段，控制文件必须是可读写的。但是其他任何用户都无法修改控制文件，只有数据库服务器可以修改控制文件中的信息。

10.2　如何获得控制文件的信息

控制文件是数据库启动时非常重要的一个文件，通常一个数据库需要至少 3 个控制文件，而且这些控制文件最好不要放在同一个磁盘上，这样可以防止磁盘故障造成数据库无法启动，那么在 Oracle 数据库上，控制文件作为物理文件到底放在什么地方呢。

在数据库启动和控制文件关系中，控制文件的位置通过参数文件获得，显然我们可以打开参数文件获得控制文件的位置。但是这种方式不方便而且不安全，如果用户不小心输入了某个字符，会造成控制文件错误。Oralce 提供了视图 v$parameter 来查看控制文件的位置，如例子 10-1 所示。

例子 10-1　使用视图 v$parameter 来查看控制文件的位置

```
SQL> SELECT value
  2  FROM v$parameter
  3  where name = 'control files';

VALUE
----------------------------------------------------------------------
C:\oracle\oradata\Lin\CONTROL01.CTL, C:\oracle\oradata\Lin\CONTROL02.CTL,
C:\oracle\oradata\Lin\CONTROL03.CTL
```

从输出可以看出，该数据库有 3 个控制文件，分别为 CONTROL01.CTL、CONTROL02.CTL、CONTROL03.CTL，它们位于同一个目录 C:\oracle\oradata\Lin 下。因为读者的数据库安装目录不同，显示的输出结果或许与上述的输出略有不同，关键是读者要知道数据字典 v$parameter 的作用。

也可以使用如下方式查看当前控制文件的位置，如例子 10-2 所示。

例子 10-2　使用 show parameter 查看当前控制文件的位置

```
SQL> show parameter control files;

NAME                        TYPE    VALUE
-------------------------------------------------------------------------
control files                     string    F:\APP\ADMINISTRATOR\ORADATA\O
                                     RCL\CONTROL01.CTL, F:\APP\ADMI
NISTRATOR\ORADATA\ORCL\CONTROL
                                     02.CTL, F:\APP\ADMINISTRATOR\O
                                     RADATA\ORCL\CONTROL03.CTL
```

使用该方式查看控制文件的位置时，默认输出 3 列，分别是参数名 NAME、参数类型 TYPE 和参数值 VALUE。显然参数值和例子 10-1 的输出结果一样。

通过数据字典 v$controlfile 也可以查看控制文件的名字和存储目录，如例子 10-3 所示。

例子 10-3　通过数据字典 v$controlfile 查看控制文件的名字和存储目录

```
SQL> col name for a50
```

```
SQL> select status , name
  2* from v$controlfile

STATUS  NAME
------- ---------------------------------------------------
        F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL01.CTL
        F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL02.CTL
        F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL03.CTL
```

在 v$parameter 视图中记录控制文件名字和目录的列名为 VALUE，而在 v$controlfile 视图中记录控制文件名字和目录的列名为 NAME。从上述 3 个示例输出可以看出它们的输出结果相同。

10.3　控制文件的内容

控制文件是二进制文件，是无法通过文本编辑器查看的，而且该文件由 Oracle 数据库服务器自动维护，DBA 无法干预。我们可以通过 Oralce 的文档得知控制文件中的内容，以及使用 v$controlfile_record_section 视图查看所有的记录信息。

10.3.1　控制文件中所存的内容

在相关文档中，说明了控制文件中存放了如下的信息。

- 数据库名：在初始化参数 DB_NAME 中获得，或是 CREATE DATABASE 语句执行时使用的名字。
- 数据库标识符：数据库创建时 Oralce 记录的标识符。
- 数据库创建时间：创建数据库时由 Oracle 自动记录。
- 表空间信息：当表增加或删除表空间时记录该信息。
- 重做日志文件历史：在日志切换时记录。
- 归档日志文件的位置和状态信息：在归档进程发生时完成。
- 备份的状态信息和位置：由恢复管理器记录。
- 当前日志序列号：日志切换时记录。
- 检验点信息：当检验点事件发生时记录。

10.3.2　如何查看控制文件中所存内容的记录信息

控制文件中到底放了什么东西，以及如何查看呢？我们通过例子 10-4 查询控制文件中所存储的内容，此时使用动态数据字典视图 v$controlfile_record_section，如例子 10-4 所示。

例子 10-4　查询控制文件中所存储的内容

```
SQL> SELECT type,record_size,records_total,records_used
  2  FROM v$controlfile_record_section;

TYPE                RECORD_SIZE RECORDS_TOTAL RECORDS_USED
---------------------------- -------------------- -------------------------
```

```
------------------------
    DATABASE                1102            1           1
    CKPT PROGRESS           2036            4           0
    REDO THREAD             104             1           1
    REDO LOG                72              5           3
    DATAFILE                180             100         10
    FILENAME                524             116         10
    TABLESPACE              68              100         11
    TEMPORARY FILENAME      56              100         0
    RMAN CONFIGURATION      1108            50          0
    LOG HISTORY             36              226         23
    OFFLINE RANGE           56              145         0

    TYPE             RECORD SIZE RECORDS TOTAL RECORDS USED
    ------------------ ----------- ------------- ------------
    ARCHIVED LOG            584             230         22
    BACKUP SET              40              101         0
    BACKUP PIECE            736             204         0
    BACKUP DATAFILE         116             210         0
    BACKUP REDOLOG          76              53          0
    DATAFILE COPY           660             203         0
    BACKUP CORRUPTION       44              185         0
    COPY CORRUPTION         40              101         0
    DELETED OBJECT          20              407         0
    PROXY COPY              852             210         0
    RESERVED4               1               4072        0

    已选择 22 行。
```

从上述输出可以看出控制文件中存放了创建数据库的信息、重做日志信息、数据文件以及归档日志文件记录等信息。

控制文件中记录了大量很有价值的信息用于数据库维护和管理，很多动态数据字典视图就是从控制文件中获得数据的，这些数据字典视图如下所示。

- v$backup
- v$database
- v$tempfile
- v$tablespace
- v$archive
- v$log
- v$logfile
- v$loghist
- v$archived_log
- v$database

下面我们说明从控制文件中获得相关数据的视图的用处，如 v$database 视图就是从控制文件中获得基础数据，通过该视图可以查看数据库 ID、创建时间和数据库是否处于归档模式等。

```
    SQL> col name for a20
```

```
SQL>select name,created,log mode
  2* from v$database

NAME                 CREATED            LOG MODE
-------------------------------------------------- ----------------------
ORCL                  10-10 月-010       ARCHIVELOG
```

虽然我们不能从控制文件中直接读取关于数据库创建的信息，但是可以间接地通过数据字典视图 v$database 来查看。上面的输出显示了当前数据库的全局名为 ORCL，创建时间为 10-10 月-010 且处于归档模式，读者的输出依据自己的数据库信息会有不同。

10.4　存储多重控制文件

正是由于控制文件（control file）的重要性，就要求控制文件不能只有一个，通常生产数据库中的控制文件要多于 3 个，并且存放在不同的磁盘上。Oracle 数据库会同时维护多个完全相同的控制文件，这也称为多重控制文件。在不同磁盘上存储多重控制文件可以避免控制文件的单点失效问题。如果一个磁盘的控制文件失效，Oracle 会自动使用参数文件中记录的其他控制文件启动数据库。

在控制文件的维护中，Oracle 会建议用户遵循一个原则，即使用多重控制文件，并将控制文件的副本存储在不同的磁盘上，监控备份工作。下面将依次讲解如何使用多重控制文件并将文件副本存储在不同的磁盘上、如何备份控制文件，以及如何恢复控制文件。

10.4.1　多重控制文件

在 Oralce 数据库中，控制文件的默认存储目录和数据库文件，重做日志文件等存放在同一个目录下，以 Oralce 11g 为例，该目录为：

```
$ORACLE_BASE\oradata\ORACLE_SID.
```

在笔者安装的 Oracle 11g 中，ORACLE_BASE 为 F:\APP\ADMINISTRATOR，而 ORACLE_SID 为 ORCL，所以控制文件目录如下：

```
F:\APP\ADMINISTRATOR\ORADATA\ORCL
```

Oralce 会同时建立 3 个控制文件，默认控制文件名依次为 CONTROL1.CTL、CONTROL2.CTL 和 CONTROL3.CTL，如例子 10-5 所示。

例子 10-5　查看当前数据库上的控制文件分布

```
SQL> col name for a50
SQL> select status,name
  2  from v$controlfile;

STATUS  NAME
------------- -----------------------------------------------------------
       F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL01.CTL
       F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL02.CTL
       F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL03.CTL
```

把这么重要的控制文件放在同一个磁盘的同一个目录下显然是不安全的，我们应该遵循 Oracle 的忠告，使用多重控制文件并存储在不同的磁盘上。

10.4.2　移动控制文件

数据库在启动时首先要读取参数文件，而参数文件有传统的 PFILE（init.ora）文件和 SPFILE 文件，针对采用不同的数据库启动初始化参数文件实现控制文件的分布式存储的方式略有不同。下面依次演示。

1. 使用 PFILE（init.ora）文件时移动控制文件

PFILE 文件是一个可识别的正文文件，我们可以对存储在 PFILE 中的参数直接更改，这就方便了使用 PFILE 实现移动控制文件的存储方式，其具体步骤如下：

01 利用数据字典获得控制文件的名字。

例子 10-6　数据字典 v$parameter 获得控制文件的名字

```
SQL> select value
  2  from v$parameter
  3  where name = 'control files';

VALUE
--------------------------------------------------------------------------
F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL01.CTL,F:\APP\ADMINISTRATOR\ORADA
TA\ORCL\CONTROL02.CTL,F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL03.CTL
```

此时输出的控制文件信息就是参数文件中保存的控制文件名及目录信息，我们也可以打开参数文件来验证，如图 10-2 所示。



图 10-2　参数文件中的控制文件信息

02 关闭数据库。

例子 10-7　关闭数据库

```
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
```

03 修改参数文件 PFILE 中参数 control_files 的值，即更改控制文件名而使得该文件存储在不

同目录下并保存该文件。修改结果如图 10-3 所示。

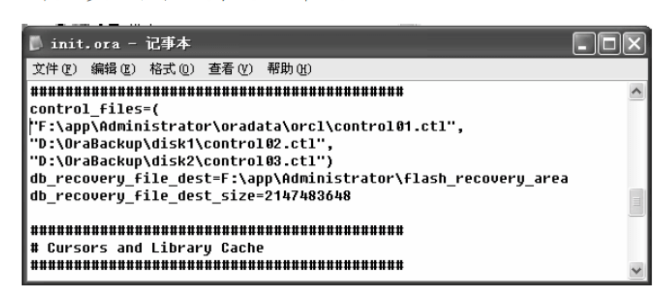


图 10-3 修改参数文件中的控制文件名

04 使用操作系统命令把步骤 1 中的控制文件 CONTROL2.CTL 和 CONTROL2.CTL 分别复制到目录 D:\OraBackup\disk1 和 D:\OraBackup\disk2 下。之后需要删除掉步骤 1 中默认目录下的 CONTROL2.CTL 和 CONTROL3.CTL 文件，以防止文件冗余。

05 重启数据库。

例子 10-8 重启数据库

```
SQL> startup
ORACLE 例程已经启动。

Total System Global Area    118255568 bytes
Fixed Size                  282576 bytes
Variable Size               83886080 bytes
Database Buffers            33554432 bytes
Redo Buffers                532480 bytes
数据库装载完毕。
数据库已经打开。
```

06 验证修改结果。

例子 10-9 验证控制文件的修改结果

```
SQL> col name for a40
SQL>SELECT *
  2* FROM v$controlfile

STATUS  NAME
------- ----------------------------------------
       F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL01.CTL
       D:\ORABACKUP\DISK1\CONTROL02.CTL
       D:\ORABACKUP\DISK2\CONTROL03.CTL
```

此时，我们的修改成功，3 个控制文件存储在不同磁盘上（CONTROL02.CTL 和 CONTROL03.CTL 读者可以理解为存储在不同的磁盘，因为笔者的计算机只有两个磁盘分区，所以在 D 盘的目录 ORABACKUP 下使用 disk1 和 disk2 模拟不同的磁盘），3 个不同磁盘上的控制文件由 Oracle 数据库服务器自动维护，一旦发生诸如增删数据文件或更改重做日志名等事件，Oracle 数据库服务器会同时更改这 3 个控制文件中的信息。

2. 使用 SPFILE 文件时移动控制文件

因为 SPFILE 是二进制文件，所以无法通过修改 PFILE 的方式修改控制文件名，Oralce 提供了 ALTER SYSTEM 指令允许修改 SPFILE 中的参数。此时实现控制文件分布式存储的步骤如下。

01 获取控制文件名，如果读者清楚自己的控制文件信息，该步骤可以省略。

例子 10-10　获取控制文件名

```
SQL> select *
  2  from v$controlfile;

STATUS  NAME
------- ----------------------------------------
        F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL01.CTL
        F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL02.CTL
        F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL03.CTL
```

02 使用 alter system set 指令修改 SPFILE 中的控制文件名。

例子 10-11　使用 alter system set 指令修改 SPFILE 中的控制文件名

```
SQL> alter system set control files =
  2  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL01.CTL ',
  3  'D:\ORABACKUP\DISK3\CONTROL02.CTL',
  4  'D:\ORABACKUP\DISK4\CONTROL03.CTL' SCOPE = SPFILE;

系统已更改。
```

03 关闭数据库。

例子 10-12　关闭数据库

```
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
```

04 将控制文件 CONTROL02.CTL 和 CONTROL03.CTL 复制到更改的目录下。即将 CONTROL02.CTL 复制到目录 D:\ORABACKUP\DISK3 下，将 CONTROL03.CTL 复制到目录 D:\ORABACKUP\DISK4 下。

05 重启数据库。

例子 10-13　正常启动数据库

```
SQL> conn /as sysdba
已连接到空闲例程。
SQL> startup
ORACLE 例程已经启动。

Total System Global Area    53108568106 bytes
Fixed Size                  1034380 bytes
Variable Size               247464852 bytes
Database Buffers            281018368 bytes
```

```
Redo Buffers                 1003102106 bytes
数据库装载完毕。
数据库已经打开。
```

我们再验证下是否使用 PFILE 启动数据库，如例子 10-14 所示。

例子 10-14　验证是否使用 PFILE 启动数据库

```
SQL> show parameter spfile;

NAME                    TYPE        VALUE
------------------------------------ ------------------------------
spfile                  string      F:\APP\ADMINISTRATOR\PRODUCT\1
                                    1.1.0\DB 1\DATABASE\SPFILEORCL
                                    .ORA
```

显然，现在 VALUE 的值不为空，说明此时使用 SPFILE 文件启动数据库。

06 验证修改结果。

例子 10-15　验证控制文件的修改结果

```
SQL> select status,name
  2  from v$controlfile;

STATUS  NAME
------- ----------------------------------------
        F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL01.CTL
        D:\ORABACKUP\DISK3\CONTROL02.CTL
        D:\ORABACKUP\DISK4\CONTROL03.CTL
```

从上述输出看出，我们已经将名为 CONTROL02 和 CONTROL03 的控制文件分布到了 D:\ORABACKUP\DISK3 和 D:\ORABACKUP\DISK4 目录下。

10.4.3　添加控制文件

Oracle 默认建立 3 个控制文件，使用 10.4.2 节中介绍的方法可以移动控制文件，将控制文件存储在不同的磁盘空间，以防止控制文件的单点失效。在生产数据库中往往至少需要 3 个控制文件，如果需要 5 个控制文件该如何处理呢？下面给出一个具体步骤，但不做详细过程演示，以数据库启动时采用 PFILE 参数文件为例。

01 查看控制文件的名字。

02 关闭数据库。

03 使用操作系统命令将步骤 1 中的一个控制文件复制到一个目录下并修改控制文件的名字，如复制到目录 D:\OraBackup\disk5 下，控制文件名修改为 CONTROL05.CTL。

04 修改参数文件中参数 control_files 的值，添加一个控制文件名，如 D:\OraBackup\disk5\CONTROL05.CTL。

05 重新启动数据库。

10.5　备份和恢复控制文件

由于控制文件在数据库启动过程中的重要性，所以最好备份控制文件，这样在发生控制文件损坏时，可使用备份的文件来恢复控制文件，保证数据库的正常启动和运行。

10.5.1　控制文件的备份

由于控制文件的重要性，备份控制文件是非常必要的，这也是 Oracle 极力推荐的，如例子 10-16 所示。

例子 10-16　使用 ALTER DATABASE BACKUP CONTROL 备份控制文件

```
SQL> alter database backup controlfile to
2  'd:\OraBackup\disk5\backup controlfile 010 06 14.ora';

数据库已更改。
```



目录 D:\OraBackup\disk5 必须是存在的，Oracle 会自动在该目录下创建一个备份文件 backup_controlfile_010_06_14.ora。由于 Oracle 数据库服务器不断地更改控制文件中的信息，所以备份的控制文件不是最新的，在恢复数据库时最好不要使用备份的控制文件，这样会造成数据丢失。

我们通过 Windows 的资源管理器来查看备份的文件是否存在，如图 10-4 所示。

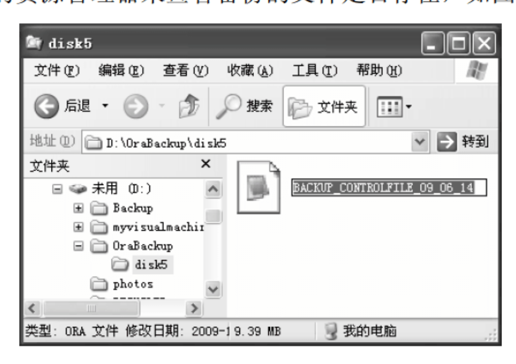


图 10-4　查看备份的控制文件

Oracle 也提供了一种方式用于备份控制文件，即将控制文件备份到追踪文件中，使用该文件就可以恢复控制文件（将在 10.5.2 节中详解）。其备份方法首先要设置参数 sql_trace 为 true，如例子 10-17 所示。

例子 10-17　设置参数 sql_trace 为 true

```
SQL> alter session set sql trace = true;

会话已更改。
```



在使用该方式备份控制文件时，必须把参数 sql_trace 的值设置为 true，Oracle 默认该参数值为 false。再使用如下指令将控制文件备份的追踪文件中。

```
SQL> alter database backup controlfile to trace;
数据库已更改。
```

Oracle 提供了一个参数 user_dump_dest，可以查看跟踪文件的存储目录。使用例子 10-18 查询该存储目录。

例子 10-18　查询参数 user_dump_dest 指定的目录

```
SQL> show parameter user_dump_dest;

NAME                          TYPE        VALUE
------------------------------------------------------------------------------
user_dump_dest                string      F:\app\administrator\diag\rdbm
                                          s\orcl\orcl\trace
```

从输出看出，跟踪文件的存储目录为 F:\app\administrator\diag\rdbms\orcl\orcl\trace，打开该目录，按照时间顺序对该目录下的文件进行排序，打开最近建立的跟踪文件，如图 10-5 所示。

```
orcl_ora_2768 - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
-- After mounting the created controlfile, the following SQL
-- statement will place the database in the appropriate
-- protection mode:
--  ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PERFORMANCE
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE "ORCL" RESETLOGS  ARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 3
    MAXDATAFILES 100
    MAXINSTANCES 8
    MAXLOGHISTORY 292
LOGFILE
  GROUP 1 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG'  SIZE 50M,
  GROUP 2 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG'  SIZE 50M,
  GROUP 3 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG'  SIZE 50M
-- STANDBY LOGFILE
DATAFILE
  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSTEM01.DBF',
  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSAUX01.DBF',
  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\UNDOTBS01.DBF',
  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS01.DBF',
  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\EXAMPLE01.DBF'
CHARACTER SET ZHS16GBK
;
-- Commands to re-create incarnation table
-- Below log names MUST be changed to existing filenames on
-- disk. Any one log file from each branch can be used to
-- re-create incarnation records.
-- ALTER DATABASE REGISTER LOGFILE
'F:\APP\ADMINISTRATOR\FLASH_RECOVERY_AREA\ORCL\ARCHIVELOG\2009_10_22\O1_MF_1_1_%U_.ARC';
-- ALTER DATABASE REGISTER LOGFILE
'F:\APP\ADMINISTRATOR\FLASH_RECOVERY_AREA\ORCL\ARCHIVELOG\2009_10_22\O1_MF_1_1_%U_.ARC';
-- Recovery is required if any of the datafiles are restored backups,
-- or if the last shutdown was not normal or immediate.
RECOVER DATABASE USING BACKUP CONTROLFILE
-- Database can now be opened zeroing the online logs.
ALTER DATABASE OPEN RESETLOGS;
-- Commands to add tempfiles to temporary tablespaces.
```

图 10-5　建立控制文件的跟踪文件内容

10.5.2　控制文件的恢复

在数据库中如果有一个或多个控制文件丢失或出错，我们可以根据不同的情况进行处理。

1. 部分控制文件损坏的情况

如果数据库正在运行，我们可以先关闭数据库，再将完好的控制文件复制到已经丢失或出错的控制文件的位置，但是要更改文件名字为该丢失或出错的控制文件的名字。如果存储丢失的控制文件的目录也被破坏，则需要重新建立一个新的目录用于存放新的控制文件，并为该控制文件命名。此时需要修改数据库初始化参数文件中控制文件的位置信息。

2. 控制文件全部丢失或损坏

此时使用备份的控制文件（这也是为什么 Oracle 强调在数据库结构发生变化后要进行控制文件备份的原因）重建控制文件。先关闭数据库，再将备份的控制文件复制到先前控制文件的所在位置上，并更改备份控制文件名为先前控制文件的文件名。

接下来打开数据库到 MOUNT 状态，如下所示。

```
SQL>startup mount
```

然后打开数据库，如下所示。

```
ALTER DATABASE OPEN USING BACKUP CONTROLFILE;
```



此时由于使用备份的控制文件所以会有数据丢失的情况，因为从该备份文件被备份时刻起到控制文件发生故障时间段内发生的数据变化无法恢复。

- 通过跟踪文件重建控制文件

跟踪文件中记录了用于建立控制文件的 SQL 语句，适当的编辑跟踪文件，然后使用 SQL 指令重建控制文件。

① 编辑跟踪文件

在图 10-5 的跟踪文件的内容中，从 STARTUP NOMOUNT 到 CHARACTER SET ZHS16GBK 的部分提取出来，再增加一些指令来制作脚本文件 createctl.sql，将下面的指令保存为 createctl.sql 脚本文件。

```
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE "ORCL" RESETLOGS  ARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 3
    MAXDATAFILES 100
    MAXINSTANCES 8
    MAXLOGHISTORY 2102
LOGFILE
  GROUP 1 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG'  SIZE 50M,
  GROUP 2 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG'  SIZE 50M,
  GROUP 3 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG'  SIZE 50M
-- STANDBY LOGFILE
DATAFILE
  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSTEM01.DBF',
  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSAUX01.DBF',
  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\UNDOTBS01.DBF',
```

```
  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS01.DBF',
  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\EXAMPLE01.DBF'
CHARACTER SET ZHS16GBK
;
RECOVER DATABASE
ALTER SYSTEM ARCHIVE LOG ALL;
ALTER DATABASE OPEN;
ALTER TABLESPACE TEMP ADD TEMPFILE ' C:\ORACLE\ORADATA\LIN \TEMP01.DBF'
    SIZE 411043040  REUSE AUTOEXTEND ON NEXT 655360  MAXSIZE 32767M;
```

② 运行该脚本文件

```
SQL> @createctl.sql
SQL> STARTUP NOMOUNT
ORACLE instance started.

Total System Global Area  105337420 bytes
Fixed Size                 452044 bytes
Variable Size            10100511004 bytes
Database Buffers          25165824 bytes
Redo Buffers               667648 bytes
SQL> CREATE CONTROLFILE REUSE DATABASE "   ORCL" NORESETLOGS  ARCHIVELOG
  2  --  SET STANDBY TO MAXIMIZE PERFORMANCE
  3      MAXLOGFILES16
  4      MAXLOGMEMBERS 3
  5      MAXDATAFILES 100
  6      MAXINSTANCES8
  7      MAXLOGHISTORY 102
  8  LOGFILE
  10  GROUP 1 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG'  SIZE 50M,
 10  GROUP 2 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG'  SIZE 50M,
 11  GROUP 3 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG'  SIZE 50M,
 12  DATAFILE
 10 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSTEM01.DBF',
14  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSAUX01.DBF',
15  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\UNDOTBS01.DBF',
16  'F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS01.DBF',
17   'F:\APP\ADMINISTRATOR\ORADATA\ORCL\EXAMPLE01.DBF'
18   CHARACTER SET ZHS16GBK
;

Control file created.

SQL> RECOVER DATABASE
ORA-00283: recovery session canceled due to errors
ORA-00264: no recovery required
SQL> ALTER SYSTEM ARCHIVE LOG ALL;

System altered.

SQL> ALTER DATABASE OPEN;

Database altered.
```

```
SQL>        ALTER        TABLESPACE        TEMP        ADD        TEMPFILE
'F:\APP\ADMINISTRATOR\ORADATA\ORCL\TEMP01.DBF'
    SIZE 28311552  REUSE AUTOEXTEND ON NEXT 655360  MAXSIZE 32767M;

Tablespace altered.

SQL>
```

在使用 TRACE 文件重新建立数据库控制文件时，通过 TRACE 文件知道该数据库的日志文件、数据文件、数据库名和其他一些参数信息，通过执行一个编辑好的脚本文件可以重新建立一个可用的控制文件。在用户的具体操作中需要依据数据库服务器本身的具体情况具体分析。

- 手工重建控制文件

在上面使用跟踪文件恢复数据库控制文件的过程中，读者已经看到了使用一个编辑好的脚本文件创建控制文件的过程，如果在脚本文件中的参数都很清楚，当然可以使用 CREATE CONTROLFILE 指令逐步创建控制文件。

10.6　本章小结

控制文件在数据库的启动过程中起着关键作用，当数据库启动时，需要通过控制文件找到数据文件、日志文件的位置，同时需要检查控制文件中记录的检查点 SCN 序列号和数据文件中的检查点序列号对比来实现数据库实例的恢复，如果数据库控制文件损坏则数据库无法启动。鉴于控制文件的重要性，Oralce 支持存储多重控制文件，并且最好把控制文件存储在不同的磁盘目录下，读者在理解了多重控制文件的概念后，进一步掌握移动控制文件和添加控制文件实现控制文件的维护。数据库控制文件的备份和控制是 DBA 需要掌握的内容，本章重点介绍了冷备份控制文件以及使用 TRACE 文件重新建立控制文件，使用冷备份会造成信息的丢失，而使用 TRACE 文件也需要做一些准备工作，至少可以找到该 TRACE 文件。当然如果读者对于自己数据库的各种文件目录和其他参数设置很清楚的话，可以使用手工重新建立数据库控制文件。

第 11 章

重做日志管理

重做日志是 Oracle 数据库中很重要的一部分内容。在数据库恢复时，需要掌握重做日志的工作原理，学会如何配置和维护重做日志。本章首先介绍 Oracle 引入重做日志的原因，介绍重做日志的工作过程。进而介绍 Oracle 的重做日志结构、重做日志组和重做日志成员。读者可以把握如何创建和删除重做日志组和重做日志成员。日志切换和检查点事件是涉及重做日志维护的重要事件，通过对这两个事件的学习有助于实现重做日志的维护。最后介绍与重做日志相关的报警文件信息和 LGWR 追踪文件，这对于故障处理提供了很好的线索。最后介绍归档重做日志。

11.1　Oracle为何引入重做日志

我们可以用 4 个字来说明 Oracle 引入重做日志的原因：数据恢复。在数据库运行过程中，用户更改的数据会暂时存放在数据库高速缓冲区中，而为了提高写数据库的速度，不是一旦有数据变化，就把变化的数据写到数据文件中，频繁的读写磁盘文件使得数据库系统效率降低，所以，要等到数据库高速缓冲区中的数据达到一定的量或者满足一定条件时，DBWR 进程才会将变化了的数据提交到数据库中，也就是 DBWR 将变化了的数据写到数据文件中。这种情况下，如果在 DBWR 把变化了的更改写到数据文件之前发生了宕机，那么数据库高速缓冲区中的数据就全部丢失，如果在数据库重新启动后无法恢复这部分用户更改的数据，显然是不合适的。

而重做日志就是把用户变化了的数据首先保存起来，其中 LGWR 进程负责把用户更改的数据优先写到重做日志文件中。在数据库原理课程中，这种机制也叫做日志写优先。这样在数据库重新启动时，数据库系统会从重做日志文件中读取这些变化了的数据，将用户更改的数据提交到数据库中，写入数据文件。

为了提高磁盘效率，并为了防止重做日志文件的损坏，Oracle 引入了一种重做日志结构，如图 11-1 所示。

在该示例图中，重做日志文件结构由 3 个重做日志组组成，每个重做日志组中有两个重做日志成员（重做日志文件），当然可以有更多的重做日志组，每个组中也可以有更多的重做日志成员。数据库系统会先使用重做日志组 1，该组写满后，就切换到重做日志组 2，再写满后，继续切换到重做日志组 3，然后循环使用重做日志组 1，Oracle 以这样循环的方式使用重做日志组。

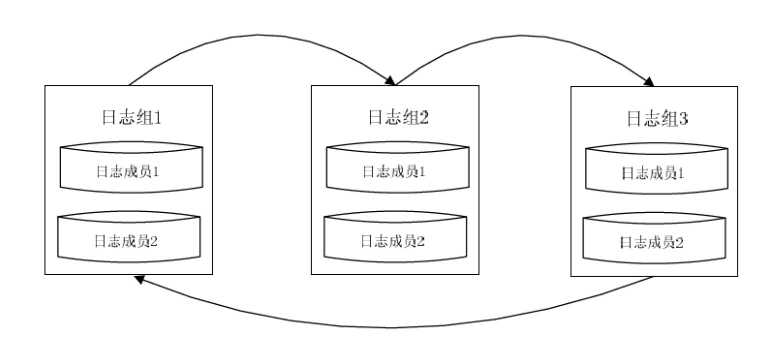


图 11-1　重做日志文件结构

该结构直观地说明重做日志文件的组成，Oracle 规定每个数据库实例至少有两个重做日志组，每个重做日志组至少有一个重做日志文件。当重做日志组中有多个日志成员时，每个重做日志成员的内容相同，Oracle 会同步同一个重做日志组中的每个成员。在工作过程中，Oracle 循环地使用重做日志组，当一个重做日志组写满时，就自动进行日志切换，切换到它可以找到的其他重做日志组，并为该日志组设置一个日志序列号。在必要的条件下也可以实现强制日志切换。

如果没有启动归档日志，当一个循环结束，再次使用先前的重做日志组时，会以覆盖的方式向该组的重做日志文件中写数据。在非归档模式下，在重新使用新的联机重做日志前，DBWR 进程需要将所有的数据更改写到数据文件中，这有时也称为 DBWR 归档。所以，对于生产数据库要求工作在归档模式下。

如果数据库处于归档模式下，当前正在使用的重做日志写满后，Oracle 会关闭当前的日志文件，ARCH 进程把旧的重做日志文件中的数据移动到归档重做日志文件中。归档完成后，寻找下一个可用重做日志组，找到该组中可用的日志文件，打开该文件并实现写操作。归档进程并不是一直存在。



如果数据库处于归档模式，在归档进程 ARCH 把联机重做日志移动到归档日志前，Oracle 无法使用一个已经关闭的重做日志。即如果 ARCH 没有完成，就没有已经归档的联机重做日志可以用于切换，只有 ARCH 释放了联机重做日志后，数据库才可以继续工作。

11.2　读取重做日志文件信息

在 11.1 节介绍了 Oracle 为何引入重做日志文件，以及重做日志文件的工作机制和文件结构，那么在一个数据库系统中如何查看关于重做日志文件的信息呢。我们通过两个动态数据字典视图 v$log 和 v$logfile 来获取关于重做日志文件的信息。

11.2.1　v$log 视图

数据字典视图 v$log 记录了当前数据库的日志组号、日志序列号、每个日志文件的大小（以字节为单位）、每个日志组的成员数量，以及日志组的当前状态。例子 11-1 为使用 v$log 查看重做

日志信息。

例子 11-1　使用 v$log 查看重做日志信息

```
SQL>conn /as sysdba
SQL> select group#,sequence#,bytes,members,archived,status
  2  from v$log;

    GROUP#  SEQUENCE#      BYTES    MEMBERS ARC STATUS
---------- ---------- ---------- ---------- --- ------------
         1        39   114115760          1 NO  CURRENT
         2        311  114115760          1 NO  INACTIVE
         3        37   114115760          1 NO  INACTIVE
```

该例子 11-1 的输出说明，当前有 3 个日志组，与每个日志文件对应的日志序列号，该序列号是全局唯一的，同一个日志组中的日志序列号相同，用户数据库恢复时使用。每个日志组的成员数量及每个日志组的当前状态。重做日志组 1 为当前正在使用的重做日志组，该日志组中有最大日志序列号，该日志文件还没有归档。

11.2.2　v$logfile 视图

数据字典视图 v$logfile 记录了当前日志组号、该日志组的状态、类型和日志组成员信息，下面用例子 11-2 说明该数据字典视图的输出。

例子 11-2　使用数据字典视图 v$logfile 查看重做日志组信息

```
SQL>conn /as sysdba
SQL> col member for a50
SQL> select group#,status,type,member
  2  from v$logfile;

    GROUP# STATUS  TYPE    MEMBER
---------- ------- ------- ------------------------------------
         3 STALE   ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG
         2 STALE   ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG
         1         ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG
```

在解释输出结果前，先介绍一下 STATUS 参数的含义：

- STALE：说明该文件内容是不完整的。
- 空白：说明该日志组正在使用。
- INVALID：表示该文件不能被访问。
- DELETED：表示该文件已经不再使用。

下面继续说明例子 11-2 的输出，我们可以知道，该数据库系统有 3 个重做日志组，每个日志组有一个重做日志成员，且都为联机（ONLILNE）重做日志文件。其实 DBA 如果看到这样的情况，应该知道需要增加重做日志成员，并且把每个日志组的重做日志成员分布在不同磁盘上。

11.2.3　判断是否归档

要查看当前的数据库是否处于归档模式，可以使用例子 11-3 查看。

例子 11-3　查看当前的数据库是否处于归档模式

```
SQL>conn /as sysdba
SQL> archive log list;
数据库日志模式                非存档模式
自动存档                      禁用
存档终点                      USE_DB_RECOVERY_FILE_DEST
最早的概要日志序列            37
下一个存档日志序列            37
当前日志序列                  39
```

上述输出说明，该数据库不处于日志归档模式，自动存档禁用。下节将介绍如何设置数据库归档模式，以及启动自动存档。

11.2.4　设置数据库为归档模式

在生产数据库中，为了防止介质失败造成的数据恢复，在 Oracle 9i 和 Oracle 11g 中，设置数据库为归档模式的方法有所不同，从这些差异中可以看出 Oracle 对于技术和应用的改进。

（1）如何在 Oracle 11g 中设置数据库为归档模式。

首先要关闭数据库，再启动数据库到 mount 状态，如例子 11-4 所示。

例子 11-4　关闭数据库并启动数据库到 mount 状态

```
QL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
RACLE 例程已经关闭。
QL> startup nomount;
RACLE 例程已经启动。

otal System Global Area      11112555611 bytes
ixed Size                    2112576 bytes
ariable Size                 113111160110 bytes
atabase Buffers              33554432 bytes
edo Buffers                  5324110 bytes
SQL> alter database mount;

数据库已更改。

SQL> alter database archivelog;

数据库已更改。
```

此时虽然数据库处于归档模式，而且是自动归档。在设置了数据库为归档模式，而且为自动归档后，我们用例子 11-5 验证修改结果。

例子 11-5　验证数据是否处于归档模式

```
SQL>conn /as sysdba
SQL> archive log list;
数据库日志模式              存档模式
自动存档                    启用
存档终点                    USE DB RECOVERY FILE DEST
最早的概要日志序列          37
下一个存档日志序列          39
当前日志序列                39
```

上例说明数据库日志模式已经处于“存档模式”，存档重点为参数 db_recovery_file_dest 指定的目录，我们可以使用如下查询知道该参数指定的文件目录。

例子 11-6　查看参数 db_recovery_file_dest 的值

```
SQL> show parameter db recovery file dest;

NAME                          TYPE          VALUE
------------------------------------------------------------------------
db recovery file dest string F:\app\Administrator\flash rec overy area
db_recovery_file_dest_size          big integer       2G
```

从参数 db_recovery_file_dest 的值为可以知道归档文件的存储目录，并且参数 db_recovery_file_dest_size 指出该目录存储文件的大小为 2G。

（2）如何在 Oracle 9i 中设置数据库为归档模式。

毕竟 Oracle 11g 是 Oracle 9i 的升级版本，在设置数据库归档模式时，步骤基本相同，首先关闭数据库，启动数据库到 mount 状态，然后使用指令 alter database archivelog 来设置自动归档，但是仍需要设置一个参数 log_archive_start=true，才可以实现数据库服务器的自动归档。

修改参数 log_archive_start 需要修改参数文件 init.ora，用记事本打开该文件，去掉参数 log_archive_start 前面的#号，然后使用该参数文件重新启动数据库，如果数据库是使用 spfile 启动的，则需要相对复杂些的步骤，即使用 create pfile from spfile，再修改 pfile 文件中参数 log_archive_start 为 true，然后再使用指令 create spfile from pfile 来重新建立 spfile，此时使用 spfile 参数文件再重新启动数据库。

11.3　重做日志组及其管理

Oracle 要求最少两个重做日志组，每个日志组至少一个日志成员，而在生产数据库中至少需要 3 个重做日志组，而每个重做日志组需要多于 3 个重做日志成员，这些日志成员最好部署在不同磁盘的不同目录下，由于重做日志文件在数据库恢复中的重要性，分布式部署的目的就是为了防止磁盘损坏造成的重做日志失效。本节讲述如何添加重做日志组及其管理。

11.3.1　添加重做日志组

向当前数据库中添加一个新的日志组的语法格式为：

```
ALTER DATABASE [database name]
        ADD LOGFILE [GROUP number]  filename SIZE n
        [,ADD LOGFILE [GROUP number]    filename SIZE n……]
```



Filename 为日志组成员的文件目录和文件名称。参数 GROUP number 可以不用，Oracle 会自动生成一个日志组号，该日志组号在原有日志组号的基础上加 1。

我们用例子 11-7 演示如何添加一个重做日志组。

例子 11-7　添加一个重做日志组

```
SQL> ALTER DATABASE ADD LOGFILE GROUP 4
  2  ('d:\temp\redo04a.log',
  3   'd:\temp\redo04b.log')
  4  SIZE 11M;

数据库已更改。
```

在例子 11-7 中 ，向当前数据库添加一个重做日志组，日志组号为 4，如果不选择 GROUP 参数，则默认在原有重做日志组号的基础上自动增长，如原来最大的日志组号为 2，则此时新建的默认组号为 3 等，依次类推。在日志组 4 中有两个日志成员，大小都为 11MB。



在例子 11-7 中，日志成员的目录必须存在，即 d:\temp 目录必须存在。

为了验证添加日志组的结果，使用例子 11-8。

例子 11-8　验证添加日志组的结果

```
SQL> select *
  2  from v$logfile;

GROUP# STATUS  TYPE    MEMBER
--------------------------------------------------------------------------
     3 STALE   ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG
     2 STALE   ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG
     1         ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG
     4         ONLINE D:\TEMP\REDO04A.LOG
     4         ONLINE D:\TEMP\REDO04B.LOG
```

例子 11-8 的输出结果说明，成功添加重做日志组，该日志组有两个日志成员，地址为 D:\TEMP\REDO04A.LOG 和 D:\TEMP\REDO04B.LOG，这两个重做日志成员都处于联机状态。

下面再添加一个重做日志组，此时不选择 GROUP 参数，向该日志组中添加 3 个日志成员，成员大小都为 11MB，如例子 11-9 所示。

例子 11-9　添加一个重做日志组并向该日志组中添加 3 个日志成员

```
SQL>  ALTER DATABASE ADD LOGFILE
  2   ('d:\disk6\redo05a.log',
  3    'd:\disk6\redo05b.log',
  4    'd:\disk6\redo05c.log')
```

```
  5  SIZE 11 M;

数据库已更改。
```

我们知道，当前的重做日志组有 4 个，例子 11-9 没有指定 GROUP 号，此时 Oracle 会为这个重做日志组自动生成一个组号，即在原有的日志组号的基础上加 1。所以新建的日志组号应该为 5，而日志成员的文件名和目录，以及文件大小如例子 11-9 中参数设置所示。

例子 11-10　验证例子 11-9 的执行结果

```
SQL>conn /as sysdba
SQL> select *
  2  from v$logfile;

GROUP# STATUS  TYPE    MEMBER
--------------------------------------------------------------------------
     3 STALE   ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG
     2 STALE   ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG
     1        ONLINE  F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG
     4        ONLINE  D:\TEMP\REDO04A.LOG
     4        ONLINE  D:\TEMP\REDO04B.LOG
     5        ONLINE  D:\DISK6\REDO05A.LOG
     5        ONLINE  D:\DISK6\REDO05B.LOG
     5        ONLINE  D:\DISK6\REDO05C.LOG
```

显然，我们新添加的重做日志组号为 5，该日志组共有 3 个重做日志文件，我们再使用例子 11-11 查询当前重做日志组的使用情况，观察新建的重做日志组的状态。

例子 11-11　查询当前重做日志组的使用情况

```
SQL> select group#,sequence#,bytes,members,archived,status
  2  from v$log;

   GROUP#  SEQUENCE#      BYTES    MEMBERS ARC  STATUS
---------- ---------- ---------- ---------- --- ----------
       1        39   114115760         1 NO   CURRENT
       2        311   114115760         1 YES INACTIVE
       3        37   114115760         1 YES INACTIVE
       4         0   114115760         2 YES UNUSED
       5         0   114115760         3 YES UNUSED
```

重做日志组 4 和重做日志组 5 是新建的重做日志组，二者的状态都为 UNSUED 未使用，重做日志组 4 有 2 个日志成员，每个成员的大小为 114 115 760 个字节，重做日志组 5 有 3 个日志成员，每个成员的大小为 114 115 760 个字节。因为两个重做日志组还没有使用所以 Oracle 没有分配日志序列号。

11.3.2　删除联机重做日志组

在不需要一个重做日志组时，可以删除该日志组。删除重做日志组的语法格式为：

```
ALTER DATABASE [database_name]
DROP LOGFILE {GROUP n|('filename'[,'filename']…)}
```

```
    [{GROUP n|('filename'[,'filename']…)}]…
```

在上述语法中符号“|”表示“或”的关系，而符号[]表示可选。我们用例子 11-12 来演示如何删除掉一个不用的重做日志组。

例子 11-12　删除重做日志组

```
SQL> alter database drop logfile group 4,group 5;

数据库已更改。
```



当前的重做日志组和处于 ACTIVE 状态的重做日志组都无法删除，如果要删除当前在用的日志组，必须先进行日志切换。在删除一个日志组时可以使用 GROUP 参数直接删除该日志组，也可以指定删除该日志组中的所有重做日志文件来达到删除日志组的效果。读者可以自己尝试这种方法。

使用例子 11-13 和例子 11-14 查询日志组 5 是否删除。

例子 11-13　验证日志组 5 的成员是否删除

```
SQL>conn /as sysdba
SQL> select *
  2  from v$logfile;

    GROUP# STATUS  TYPE    MEMBER
---------- ------- ------- -----------------------------------
         3 STALE   ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG
         2 STALE   ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG
         1          ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG
```

例子 11-14　验证日志组 5 是否删除

```
SQL> select group#,sequence#,bytes,members,archived,status
  2  from v$log;

    GROUP#  SEQUENCE#      BYTES    MEMBERS ARC  STATUS
---------- ---------- ---------- ---------- --- -----------
         1        39   114115760          1 NO   CURRENT
         2        311   114115760          1 YES  INACTIVE
         3        37   114115760          1 YES  INACTIVE
```

例子 11-13 和例子 11-14 都说明数据库系统已经将重做日志组 4 和重做日志组 5 删除。但是需要读者注意的是使用指令删除重做日志组会留下垃圾文件，也就是说在删除了重做日志组后，作为重做日志组成员的操作系统文件还存在，如图 11-2 所示。



图 11-2　删除重做日志组后的垃圾文件

所以还必须使用操作系统指令，手工删除掉这些垃圾文件。

11.4　重做日志成员及维护

重做日志成员是与重做日志组相对应的一个概念，在一个重做日志组中允许有一个重做日志成员，但是在生产数据库中要求多于 4 个日志文件，并且同一个日志组中的多个重做日志文件分布在不同磁盘的不同目录下。在同一个日志组中，日志成员的大小相同。本节讲解如何添加和删除重做日志成员，以及重做日志成员的维护。

11.4.1　添加重做日志成员

在每个重做日志组中至少要有一个日志成员，但是为了防止单点失效的发生，最好多设置几个重做日志成员，并存储在不同的磁盘空间中。这样的冗余设置可以极大地提高重做日志文件的可靠性。

向一个重做日志组中添加日志成员的语法格式如下所示。

```
ALTER DATABASE [databasename]
ADD LOGFILE MEMBER
        [‘filename’ [REUSE]
            [,’filename’[REUSE]]……
TO {GROUP n
    |(‘filename’[,’filename’]……)}
]……
```

我们用例子 11-15 来演示如何向重做日志组中添加重做日志成员，此时无论是否是当前正在使用的重做日志组，都可以添加重做日志成员，该例子中分别向重做日志 1、2、3 添加一个重做日志成员。

例子 11-15　向重做日志 1、2、3 添加一个重做日志成员

```
SQL> alter database add logfile member
  2  'd:\temp\redo01a.log' to group 1,
  3  'd:\temp\redo02a.log' to group 2,
  4  'd:\temp\redo03a.log' to group 3;

数据库已更改。
```

在成功添加后，我们用例子 11-16 验证添加结果。

例子 11-16　验证日志组的成员数结果

```
SQL> select group#,sequence#,bytes,members,archived,status
  2  from v$log;

    GROUP#  SEQUENCE#      BYTES    MEMBERS ARC  STATUS
---------- ---------- ---------- ---------- --- ------------
         1         39   114115760          2 NO  CURRENT
         2        311    114115760          2 YES INACTIVE
         3         37   114115760          2 YES INACTIVE
         4          0   114115760          2 YES UNUSED
```

在上述输出中，重做日志组 1、2 和 3 的 MEMBERS 都为 2，说明这些重做日志组有 2 个重做日志成员。再通过例子 11-17 验证添加的重做日志文件信息。

例子 11-17　验证添加的重做日志组以及对应成员信息

```
SQL> select *
  2  from v$logfile
  3  order by group#;

    GROUP# STATUS  TYPE    MEMBER
---------- ------- ------- -----------------------------------
         1         ONLINE F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG
         1 INVALID ONLINE  D:\TEMP\REDO01A.LOG
         2 STALE   ONLINE  F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG
         2 INVALID ONLINE  D:\TEMP\REDO02A.LOG
         3 STALE   ONLINE  F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG
         3 INVALID ONLINE  D:\TEMP\REDO03A.LOG
         4         ONLINE  D:\TEMP\REDO04A.LOG
         4         ONLINE  D:\TEMP\REDO04B.LOG
```

例子 11-17 中，我们使用了 order by 子句对输出进行排序，这样就方便查看每一个重做日志组的成员。在上述输出中，重做日志组 1、2 和 3 都新增了一个重做日志成员。



如果添加的日志成员文件已经存在，则需要使用 REUSE 参数，并且日志成员要用全目录格式，不要使用相对目录的形式，否则，Oracle 数据库服务器会在默认路径下建立该重做日志文件。

11.4.2　删除联机重做日志成员

如果不需要一个重做日志成员，可以删除掉，通常我们所重做日志维护就是删除和重建重做日志的过程，对于一个损坏的重做日志，如果没有及时发现使得日志切换时无法成功，则数据库最终会挂起，当然如果读者对于重做日志成员做了很好的分布存储，出现这种情况的可能性很小。但是，一旦出现重做日志文件受损的情况就要及时修复，也就是删除掉该文件，然后重建。

删除重做日志文件的语法格式为：

```
ALTER DATABASE [database name]
DROP LOGFILE MEMBER 'filename' [,'filename']……
```

下面用例子 11-18 演示如何删除一个重做日志成员。此时，只需要知道重做日志成员的目录和文件名，而不必知道该日志成员所属的日志组。在该例子中，我们删除掉重做日志组 4 中的一个日志成员。

例子 11-18　删除重做日志组中的一个日志成员

```
SQL> alter database drop logfile member 'D:\TEMP\REDO04A.LOG';

数据库已更改。
```

我通过例子 11-19 查询重做日志组 4 的日志成员信息。

例子 11-19　查询重做日志组 4 的日志成员信息

```
SQL> select *
  2  from v$logfile
  3  where group# = 4;

    GROUP# STATUS  TYPE    MEMBER
---------- ------- ------- --------------------
         4         ONLINE  D:\TEMP\REDO04B.LOG
```

输出结果说明，已经成功删除了重做日志组 4 的一个成员 D:\TEMP\REDO04A.LOG。但是操作系统中和该成员对应的文件还没有被删除，需要手动删除。

在删除日志成员时，并不是所有的重做日志成员都可以删除，Oracle 有一些限制条件。执行删除操作的一些限制如下：

- 如果要删除的日志成员是重做日志组中最后一个有效的成员，则不能删除，如该日志组中只有一个日志成员。
- 如果该日志组当前正在使用，在日志切换前不能删该组中的成员。
- 如果数据库正运行在 ARCHIVELOG 模式，并且要删除的日志成员所属的日志组没有被归档，该组中的日志成员不能被删除。

11.4.3　重设联机重做日志的大小

Oracle 没有提供直接修改联机重做日志大小的方法，而是需要先删除该日志文件所在的日志组，然后再重建日志文件，通过这种方式间接地设置联机重做日志的大小，而对于如何向重做日志组中添加日志成员和删除重做日志组在 11.4.1 和 11.4.2 节已经讲过了，

我们给出一个例子说明重设联机重做日志的大小。在该例子中，我们需要重新设置重做日志组中日志成员的大小，将其修改为 50MB，我们按照如下的步骤实现。

删除重做日志组，在这里我们假设用户的数据库上已经创建了一个重做日志组 4（如何创建重建日志组可参见 11.3 节），该日志组中有两个日志成员，分别为 E:\REDO1.LOG 和 E:\REDO2.LOG。当前的日志文件信息如例子 11-20 所示。

例子 11-20　查询当前的重做日志组信息

```
SQL> select group#,sequence#,bytes,members,status
  2  from v$log;
```

```
    GROUP#  SEQUENCE#      BYTES    MEMBERS  STATUS
---------- ---------- ---------- ---------- -----------
         1     119063   11411576          1  ACTIVE
         2     119064   11411576          1  CURRENT
         3     119062   11411576          1  INACTIVE
         4          0  157211640          2  UNUSED
```

该输出可以看出重做日志组 4 有两个成员，且大小都为 15MB。再使用例子 11-21 查看日志文件信息。

例子 11-21　查看日志组成员信息

```
SQL> select *
  2* from v$logfile

    GROUP# STATUS  MEMBER
---------- ------- ------------------------------------
         1         F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG
         2         F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG
         3         F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG
         4         E:\REDO1.LOG
         4         E:\REDO2.LOG
```

从该输出中知道日志组 4 的两个日志成员分别为 E:\REDO1.LOG 和 E:\REDO2.LOG，现在要修改重做日志组 4 中日志成员的大小，修改为 50MB。所以先删除该重做日志组，如例子 11-22 所示。

例子 11-22　删除该重做日志组

```
SQL> alter database drop logfile group 4;
alter database drop logfile group 4
*
ERROR 位于第 1 行:
ORA-01123: 日志 4 是线程 1 的当前日志 - 无法删除
ORA-00312: 联机日志 4 线程 1: 'E:\REDO2.LOG'
ORA-00312: 联机日志 4 线程 1: 'E:\REDO1.LOG'
```

但是上述输出出现问题，上述错误说明日志组 4 为当前重做日志组，数据库服务器正在使用该重做日志组，显然这样的日志组是不能删除的。为了验证我们的判断，通过例子 11-23 查看当前的重做日志组。

例子 11-23　查看当前的重做日志组

```
SQL> select group#,bytes,members,status
  2  from v$log;

    GROUP#      BYTES    MEMBERS  STATUS
---------- ---------- ---------- ----------
         1   11411576          1  INACTIVE
         2   11411576          1  ACTIVE
         3   11411576          1  INACTIVE
         4  157211640          2  CURRENT
```

上述输出验证了我们的判断，日志组 4 为当前（CURRENT）数据库服务器正在使用的重做日志组，在删除该日志组前需要使用日志切换口令，切换到其他可用的重做日志组，如例子 11-24 所示。

例子 11-24　日志切换并删除当前没使用的日志组

```
SQL> alter system switch logfile;

系统已更改。
SQL> select group#,bytes,members,status
  2  from v$log;

    GROUP#      BYTES    MEMBERS  STATUS
---------- ---------- ---------- ---------
         1   11411576          1  CURRENT
         2   11411576          1  INACTIVE
         3   11411576          1  ACTIVE
         4  157211640          2  INACTIVE
```

此时，重做日志组 4 的状态为 INACTIVE，如果在使用日志切换指令后，重做日志组 4 处于 ACTIVE 状态，则可以继续使用日志切换指令加速重做日志组 4 的归档工作，或者此时强制加入一个检查点，其指令格式为 ALTER DATABASE CHECKPOINT，使得要删除的重做日志组处于 INACTIVE 状态，因为强制检查点使得 DBWR 将保存在联机重做日志中已经变化的内容写到数据文件中。现在可以删除该重做日志组了，如例子 11-25 所示。

例子 11-25　删除重做日志组

```
SQL> alter database drop logfile group 4;

数据库已更改。
```

我们在通过例子 11-26 验证删除结果。

例子 11-26　验证例子 11-25 的删除结果

```
SQL> select *
  2  from v$logfile;

    GROUP# STATUS  MEMBER
---------- ------- -----------------------------------
         1         F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG
         2         F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG
         3         F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG
```

输出结果说明该重做日志组 4 被成功删除，下面重建该重做日志组，并注意修改日志成员的大小，如果日志成员的存储目录和文件名和先前删除的重做日志组 4 中日志成员相同，则必须先用操作系统命令删除掉重做日志 4 中的垃圾文件，否则在重新建立重做日志组 4 时，无法创建重做日志成员。下面进行第二步，重建重做日志组。

重建重做日志组 4，并修改重做日志文件的大小为 50M，如例子 11-27 所示。

例子 11-27　重建重做日志组 4 并修改重做日志文件的大小

```
SQL> alter database add logfile group 4
  2  ('e:\redo1.log',
```

```
  3  'e:\redo2.log')
  4* size 50M

数据库已更改。
```

例子 11-28　验证是否添加成功，以及日志成员的大小

```
SQL> select group#,bytes,members,status
  2  from v$log;

    GROUP#      BYTES    MEMBERS STATUS
---------- ---------- ---------- ---------
         1   11411576          1 ACTIVE
         2   11411576          1 CURRENT
         3   11411576          1 INACTIVE
         4  5242111100          2 UNUSED
```

从上述输出可以看出，日志组 4 为新建的重做日志，因为其状态信息为 UNUSED，该日志组有两个日志成员，每个日志成员的大小都为 50M。此时，成功修改了重做日志组中日志成员的大小。

11.5　清除联机重做日志

在数据库服务器处于归档（ARCHIVELOG）模式时，如果当前正在使用的重做日志组中的重做日志文件损坏，则该重做日志不能完成归档，使得数据库因无法归档而挂起，在这种情况下，需要通过清除联系机重做日志来重新初始化联机重做日志文件，如下所示。

```
ALTER DATABASE CLEAR LOGFILE GROUP n
```

在执行上面的指令后，会清除掉日志文件。



在使用清除联机重做日志文件指令后，已经删除的重做日志组中重做日志的序列号变为 0，所以此时需要做数据库的全备份，因为 Oracle 在进行数据库恢复时，需要连续的序列号。

11.6　日志切换和检查点事件

当一组重做日志组写满时，或用户发出 alter database switch logfile 时，就会触发日志切换，此时 Oracle 寻找下一个可用的重做日志组，如果数据库处于归档模式，则在将当前写满的日志组归档完成前不会使用新的重做日志组。

检查点事件是 Oracle 为了减少数据库实例恢复时间而设置的一个事件，当该事件发生时，LGWR 进程将重做日志缓冲区中的数据写入重做日志文件中，而同时通知 DBWR 进程将数据库高速缓存中的已经提交的数据写入数据文件，所以检查点事件越频繁则用于数据库恢复的重做数据就越少。此时，检验点事件也会修改数据文件头信息和控制文件信息以记录检查点的 SCN。

可以使用如下指令强制启动检查点事件。

```
alter database checkpoint
```

我们给出一个例子，先更改强制日志切换，为了加速日志切换时间，使得当前的重做日志文件处于 INACTIVE 状态，再强制产生检查点事件。

例子 11-29　强制日志切换并强制产生检查点事件

```
SQL>conn /as sysdba
SQL> alter system switch logfile;

系统已更改。
SQL> alter system checkpoint;

系统已更改。
```



检查点事件不是检验点进程触发的，如果不是强制产生检验点事件，则检验点事件由 DBWR 数据库写进程触发。

11.7　归档重做日志

归档重做日志就是联机重做日志的脱机备份，在数据库服务器处于归档模式时，发生日志切换时，数据库的归档进程 ARCH 把重做日志文件中的数据移动到归档重做日志中。归档进程在数据库服务器运行期间并不是总是存在的，而是当满足一定条件（如一组重做日志文件写满）时启动归档进程。一旦归档完毕，归档进程自动关闭。

归档日志文件存储在参数文件 SPFILE 或 init.ora 文件中参数指定的位置，在 inti.ora 文件中该参数为 log_archive_dest_n。Oracle 只能把重做日志中的数据移动到磁盘上，而不能移动到磁带等存储介质上。

11.8　本章小结

重做日志是为了数据库恢复而引入的非常重要的概念，在生产数据库中由于数据库本身崩溃或者操作系统故障都会引起数据丢失，而重做日志的存在使得在发生故障时，不会丢失用户提交的数据而实现数据库恢复或实例恢复，重做日志文件用于数据库的实例恢复，而归档重做日志用于数据库的介质恢复。

本节主要讲解了 Oracle 引入重做日志的目的，以及 Oracle 复杂的重做日志文件结构，这里主要涉及两个概念即重做日志组和重做日志成员。在理解了上述内容后，读者需要掌握如何管理重做日志组和维护重做日志文件，这是 DBA 经常使用的维护操作。日志切换和检查点事件是与重做日志文件管理相关的很重要的两个概念。需要读者认真体会，归档日志文件是重做日志文件的脱机备份，在生产数据库中要求数据库服务器处于归档模式，而设置归档模式在 Oracle 9i 和 Oracle 11g 中略有不同。最后本节给出了与联机重做日志相关的故障。

第 12 章

管理归档日志

在生产数据库中，数据库可以运行在两种模式下，一种是非归档模式，一种是归档模式。而归档直观讲就是重做日志的备份。在重要的数据库信息系统中，都将数据库运行在归档模式，管理好归档日志。这样在具有备份的情况下就可以实现数据库完全恢复。保证数据不会丢失。本章我们介绍归档模式以及在管理归档日志时需要注意的问题。

12.1 归档模式

归档模式的好处是保证数据库发生介质故障时可以完全恢复数据库，虽然介质故障的现象不同，恢复的方法也有差异，但是基本原理就是使用备份的数据和归档日志实现数据库的完全恢复。下面我们再回忆重做日志以及归档的工作过程，如图 12-1 所示。

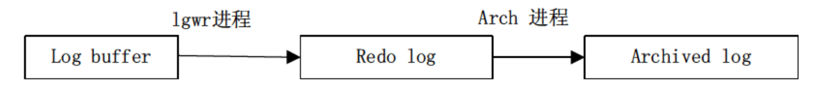


图 12-1 重做日志以及归档的工作过程

LGWR 进程将日志缓冲区中的数据写入重做日志，当前重做日志组写满后，会继续切换到新的日志组，在使用新的重做日志组之前会将当前写满的日志数据写入归档日志，注意归档进程只是在归档时才会启动。一旦当前的日志组中的重做数据都写入了归档日志，则可以切换到新的日志组并使用这个日志组。当重做日志组切换一个周期后，会覆盖已经有重做数据的日志组，但是此时已经不重要了，因为这些旧的日志数据已经写入了归档日志。这样就可以保证在数据库发生介质故障时，使用归档日志中的数据实现用户数据的不丢失恢复，即完全恢复。下面我们查看当前数据库是否处于归档模式。

例子 12-1 查看数据库的归档模式

```
SQL> archive log list;
数据库日志模式              非存档模式
自动存档                    禁用
存档终点                    USE DB RECOVERY FILE DEST
最早的联机日志序列          125
当前日志序列                127
```

这是典型的输出内容，在安装数据库时我们选择了不使用归档模式，所以此时数据库日志模

式显示为“非存档模式”，自动存档也是“禁用”，但是默认的归档终点还是有的，这个归档终点就是参数 USE_DB_RECOVERY_FILE_DEST 指定的目录。下面我们先查询下该参数默认的绝对目录是什么。

例子 12-2　查询默认归档目录

```
    SQL> show parameter db_recovery_file_dest;

    NAME                                TYPE       VALUE
    --------------------------------------------------------  -----------------
--------------------------------------------------
    db_recovery_file_dest string        F:\app\oracle\flash_recovery_area
    db_recovery_file_dest_size            big integer   3852M
```

我们的查询结果有两个参数，这个两个参数是密切相关的。一个是默认的归档日志存储的绝对路径，一个是该目录下的存储空间大小。当然这是 Oracle 默认的归档目录，我们仍然可以定义启动归档目录并方便地管理这些归档文件，即配置 log_archive_dest_n 参数，这里的 n 号代表 1~30 的整数，可以配置 30 个归档终点。并且使用 log_archive_dest_state_n 参数设置这些归档终点的状态，以便于维护。

12.2　设置归档模式

Oracle 数据库默认处于非归档模式。如果需要将数据库修改为归档模式，则需要重启数据库并进行设置。下面是具体步骤。

01 使用 SYSDBA 登录数据库。

```
C:\Documents and Settings\oracle>sqlplus /nolog

SQL*Plus: Release 11.2.0.1.0 Production on 星期一 1 月 28 12:56:30 2013

Copyright (c) 1982, 2012, Oracle.  All rights reserved.

SQL> connect sys/oracle1234 as sysdba
已连接。
```

02 使用 archive log list 指令查看当前的归档模式。

```
SQL> connect sys/oracle1234 as sysdba
已连接。
SQL> archive log list;
数据库日志模式             非存档模式
自动存档              禁用
存档终点             USE_DB_RECOVERY_FILE_DEST
最早的联机日志序列        125
当前日志序列           127
```

03 正常关闭数据库并启动数据库到 MOUNT 状态。

```
SQL> shutdown immediate;
```

```
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area  535662592 bytes
Fixed Size                  1375792 bytes
Variable Size             314573264 bytes
Database Buffers          213909504 bytes
Redo Buffers                5804032 bytes
数据库装载完毕。
```

04 使用 alter database archivelog 指令将数据库设置为归档模式，并验证数据库的归档模式是否改变。

```
SQL> alter database archivelog;

数据库已更改。

SQL> archive log list;
数据库日志模式            存档模式
自动存档             启用
存档终点            USE_DB_RECOVERY_FILE_DEST
最早的联机日志序列     125
下一个存档日志序列    127
当前日志序列           127
```

上述输出说明，数据库已经处于归档模式，使用默认的归档目录作为存储目录。此时做一次数据库的全备份，包括数据文件和控制文件，因为在之前的非归档模式下的数据库备份已经不能使用了。

一旦启动归档模式，Oracle 11g 版本的数据库会自动启动归档进程，而在之前的版本是不行的，需要手动启动这个进程，请读者注意这个问题。

12.3　设置归档进程与归档目录

在图 12-1 中，我们已经详细分析了重做日志工作的过程，以及归档工作时机。LGWR 进程将内存中的重做数据写入重做日志文件，即磁盘文件。这个过程很快，因为是从内存读数据并且与重做日志文件的格式有关。而归档进程是从重做日志文件读数据同时写入归档日志文件，这是一个读磁盘写磁盘的过程，显然这个过程相对 LGWR 的工作过程要慢一些。所以这里就会存在一个速度匹配问题。如果当前的生产数据库的 DML 操作很频繁，产生重做数据的效率很高，有可能出现归档进程无法匹配 LGWR 进程的工作，即归档进程慢。造成写重做日志的等待事件。

为了防止由于归档进程与 LGWR 进程不匹配造成的等待事件，我们可以设置一个参数，以启动更多的后台归档进程，这个参数是 log_archive_max_processes。下面我们先看一下该参数的默认值是多大。

例子 12-3　查看参数 log_archive_max_processes 值

```
SQL> show parameter log archive max processes;

NAME                                TYPE        VALUE
----------------------------------------------------- ---------------
log_archive_max_processes          integer      2
```

通过上述查询知道，该参数的默认值是 2。因为这个参数是动态参数，可以动态修改这个参数的值，如下所示。

```
SQL> alter system set log_archive_max_processes=4;

系统已更改。
```

然后我们查询该参数的值是否成功修改，如下所示。

```
SQL> show parameter log_archive_max_processes;

NAME                                TYPE        VALUE
--------------------------------------------------------------------------
log_archive_max_processes          integer      4
```

现在我们成功修改了归档进程相关的参数，以保证归档进程的速度跟得上 LGWR 的速度，防止由于写重做日志等待而造成的数据库挂起问题。

归档日志是实现数据库完全恢复的关键数据，必须做好归档日志的保护工作，Oracle 提供了 30 个归档目录，来冗余归档日志备份。只要我们实现了归档日志的物理冗余备份，就极大地保证了归档日志的高可用性，防止发生数据库介质故障造成的数据丢失。下面我们查询设置归档目录的参数。

例子 12-4　查询归档目录相关参数

```
SQL> show parameter log_archive_dest

NAME                                TYPE        VALUE
------------------------------------ ----------- ----------------
log_archive_dest                     string
log_archive_dest_1                   string
log archive dest 2                   string
log archive dest 3                   string
log archive dest 4                   string
log_archive_dest_5                   string
log_archive_dest_6                   string
log archive dest 7                   string
log archive dest 8                   string
log archive dest 9                   string
...........
log archive dest 30                  string
log archive dest state 1              string      enable

NAME                                TYPE        VALUE
------------------------------------ ----------- ----------------
```

```
log_archive_dest_state_1            string     enable
log_archive_dest_state_2            string     enable
log_archive_dest_state_3            string     enable
log_archive_dest_state_4            string     enable
log_archive_dest_state_5            string     enable
log_archive_dest_state_6            string     enable
log_archive_dest_state_7            string     enable
log_archive_dest_state_8            string     enable
log_archive_dest_state_9            string     enable
............
log_archive_dest_state_30           string     enable
```

其中 log_archive_dest_n 参数指定归档目录的绝对路径，log_archive_dest__state_n 参数指定了这些归档目录的状态，可以用于归档目录的维护。从输出我们知道这些目录默认情况下都是空的，没有设置，但是其状态都是 enable 的。在设置归档目录之前，我们先来熟悉与参数 log_archive_dest_n 设置相关的几个关键字，如表 12-1 所示。

表 12-1　相关关键字及含义

关键字	含义	例子
LOCATION	本地文件系统或 ASM 磁盘组	LOG_ARCHIVE_DEST_n= 'LOCATION=/disk1/arc' LOG_ARCHIVE_DEST_n= 'LOCATION=+DGP'
LOCATION	使用快速恢复区 Fast Recovery Area	LOG_ARCHIVE_DEST_n= 'LOCATION=USE_DB_RECOVERY_FILE_DEST'
SERVICE	通过 OracleNet ServiceName 的远程归档	LOG_ARCHIVE_DEST_n = 'SERVICE=std1'

下面我们学习如何设置这些目录。

例子 12-5　设置参数 log_archive_dest_1

```
SQL>  alter  system  set  log_archive_dest_1='location=f:\app\archive1\
mandatory';

系统已更改。
```

这里的 location 参数说明这个归档目录是在本地磁盘上，在配置 Data Guard 时，需要将归档日志发送到远端的备库，此时会使用 service 参数，这里就不在介绍。读者知道即可。并且使用了参数 mandatory，即强制该目录下的归档日志写成功。否则对应的重做日志文件不能被重新使用。下面我们查询修改结果。

例子 12-6　查询归档目录

```
SQL> show parameter log_archive_dest_1

NAME                           TYPE        VALUE
---------------------------- ------------------------------------------
log_archive_dest_1           string     location=f:\app\archive1\  mandatory
```

下面我们继续设置参数 log_archive_dest_2。

```
    SQL>  alter  system  set  log archive dest 2='location=f:\app\archive2\
optional';

    系统已更改。
```

在这个例子中，我们使用了参数 optional。意思是即使该归档目录对应的重做日志没有向该目录写成功，所对应的重做日志文件依然可以重新使用。接着我们继续设置第三个归档日志目录 log_archive_dest_3，如下所示。

```
    SQL> alter system set log archive dest 3='location=f:\app\archive3\ ';

    系统已更改。
```

我们此时没有使用任何参数，即没有 mandatory 也没有 optional，其实此时默认使用的是 optional 参数，我们通过下面的查询验证。

例子 12-7　验证归档终点设置

```
    SQL>  select  destination,binding,target,status  from  v$archive_dest  where
destination is not null;

    DESTINATION                     BINDING    TARGET     STATUS
    ---------------------------------------------------------------- --------
    f:\app\archive1\                MANDATORY  PRIMARY    VALID
    f:\app\archive2\                OPTIONAL   PRIMARY    VALID
    f:\app\archive3\                OPTIONAL   PRIMARY    VALID
```

从上面的输出可知归档目录 2、3 的 BINDING 都为 OPTIONAL。现在我们成功设置了 Oracle 的三个归档终点并设置了对应的参数，要求其中第一个归档终点必须写成功，而其他两个则不需要，这里的写成功的含义是相应的重做日志数据必须写入这个归档目录，否则对应的归档日志组就不能被重用，此时会发生数据库挂起的现象。我们再次查询此时的归档信息，如例子 12-8 所示。

例子 12-8　查询归档信息

```
    SQL> archive log list;
    数据库日志模式              存档模式
    自动存档                    启用
    存档终点                    f:\app\archive3\
    最早的联机日志序列          125
    下一个存档日志序列          127
    当前日志序列                127
```

我们看到此时的归档终点已经修改为参数 log_archive_dest_3 所设置的绝对路径。当前的归档日志序列号为 127，显然这个日志就是下一个要归档的日志。

在上面的例子中，我们设置了归档终点，并将第一个归档目录设置了参数 mandatory，即强制这个目录必须归档成功，否则未成功归档的重做日志组不能重用，数据库会挂起。在 Windows 环境下，如果修改归档参数 log_archive_dest_1 目录 F:\app\archive1\为 F:\app\archive，即 F:\app 目录存在而该目录下的 archive1\目录不存在，此时 Oracle 默认归档参数 log_archive_dest_1 目录 F:\app\archive1 的上层目录（即 F:\app）下存储归档文件。如果整个磁盘损坏无法归档成功，最终会导致数据库挂起。

为了保证归档文件高可用性，我们依然需要使用冗余的方法实现归档数据的保护，同时 Oracle 提供了参数 log_archive_min_succeed_dest，以保证最少的成功归档终点。即如果设置了 3 个归档目录，而参数 log_archive_min_succeed_dest 设置为 2，则至少保证其中两个归档终点必须归档成功。当然，此时的归档目录数大于等于参数 log_archive_min_succeed_dest 设置的值。否则会报错，错误如下所示。

```
SQL> alter system set log_archive_min_succeed_dest=4;
alter system set log archive min succeed dest=4
*
第 1 行出现错误：
ORA-02097：无法修改参数，因为指定的值无效
ORA-16020：可用的目标少于由 LOG_ARCHIVE_MIN_SUCCEED_DEST 指定的数量
```

下面我们查询参数 log_archive_min_succeed_dest 的默认值。

```
SQL> show parameter log_archive_min_succeed_dest

NAME                            TYPE       VALUE
------------------------------------------------------------------------
log_archive_min_succeed_dest    integer      1
```

参数 log_archive_min_succeed_dest 是一个动态参数，可以直接在数据库运行时修改，并且立即生效。下例修改参数的值为 2。

```
SQL> alter system set log_archive_min_succeed_dest=2;

系统已更改。
```

到现在为止我们设置了 3 个归档目录，并且强制一个归档目录必须归档成功，同时设置了参数 log_archive_min_succeed_dest 的值为 2，保证至少两个归档目录归档成功。

下面我们修改 log_archive_dest_n 参数，试图减少两个归档终点。

例子 12-9　修改 log_archive_dest_n 参数

```
SQL> alter system set log archive dest 2='';

系统已更改。
SQL> alter system set log archive dest 3='';
alter system set log archive dest 3=''
*
第 1 行出现错误：
ORA-02097：无法修改参数，因为指定的值无效
ORA-16028：新 LOG ARCHIVE DEST 3 导致少于 LOG ARCHIVE MIN SUCCEED DEST
所需的目的地数量
```

此时报错提示归档终点数与 log_archive_min_succeed_dest 参数的值相违背。必须保证有效的归档目录数不能小于 log_archive_min_succeed_dest 参数的值。

正是由于参数 log_archive_min_succeed_dest 的存在，无论我们是否设置归档目录的参数为 mandatory，其默认值都为 1，保证至少有一个归档目录必须归档成功。为了更好地理解 log_archive_min_succeed_dest 参数与 log_archive_dest_n 参数的关系，我们给出两个例子帮助读者

理解。

第一个例子。我们设置了两个归档目录，设置了参数 log_archive_dest_1 和参数 log_archvie_dest_2，并且都为 optional。设置不同的 log_archive_min_succeed_dest 参数值，从而理解 log_archive_min_succeed_dest 参数与 log_archive_dest_n 参数的关系，如表 12-2 所示。

表 12-2　参数 log_archive_min_succeed_dest 参数值设置案例 1

参数值	关系
1	至少一个 optional 的归档终点归档成功，相关重做日志才能重用
2	必须全部 optional 的归档终点归档成功，相关重做日志才能重用
>=3	该值大于 optional 的归档终点设置数量。报错

第二个例子。我们设置两个 mandatory 归档目录，3 个 optional 归档目录。设置不同的 log_archive_min_succeed_dest 参数，比较 log_archive_min_succeed_dest 参数与 log_archive_dest_n 参数的关系，如表 12-3 所示。

表 12-3　参数 log_archive_min_succeed_dest 参数值设置案例 2

参数值	关系
1	该参数值被忽略，使用设置了参数 mandatory 的归档终点数
2	Optional 的归档终点对归档成功与否没有影响，相关重做日志才能重用，此时参数 mandatory 的归档终点数与参数 log_archive_min_succeed_dest 的值相同
3	至少一个 optional 的归档终点归档成功，相关重做日志才能重用
4	至少两个 optional 的归档终点归档成功，相关重做日志才能重用
5	必须全部 optional 的归档终点归档成功，相关重做日志才能重用
>=6	报错，该参数值大于归档终点数

对于归档目录还有两个参数需要注意：一个是 log_archive_dest，一个是 log_archive_duplex_dest。如果读者只需要一个归档目录可以只设置参数 log_archive_dest，如果需要设置主备两个归档目录，可以设置 log_archive_dest 和 log_archive_duplex_dest。

使用参数 log_archive_format 可以设置归档文件的格式。下面我们查询该参数的默认值。

例子 12-10　查询参数 log_archive_format 的默认值

```
SQL> show parameter log_archive_format

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
log_archive_format                   string      ARC%S_%R.%T
```

下面我们解释参数值 ARC%S_%R.%T 的含义，其中%S 为日志序列号。%R 为 resetlogs ID，而%T 为线程号。如 ARC0000000128_0801215126.0001 说明当前的归档日志的序列号为 128，resetlogs ID 为 0801215126，线程号为 0001，对于单实例数据库，这个线程号不会改变。

参数 log_archive_format 的值可以修改，并且该参数的默认值根据平台不同具体格式也有区别。如果读者没有特殊需要采用默认值即可满足需要。



无论使用什么方式设置归档目录，归档目录所在的磁盘空间必须充分，否则归档空间不足会造成十分严重的错误，此时数据库没有任何响应，系统无法使用，并且会在告警日志中显示该错误，所以 DBA 必须监控归档空间的使用情况，防止归档空间不足造成的错误。

12.4　维护归档目录

归档目录的维护就是当归档目录所在磁盘发生故障时如何处理，这里我们先看归档目录的几个状态，即 log_archive_dest_state_n 参数的值。

- ENABLE：磁盘目录或者服务名已经指定，并且是有效的。
- DEFER：该目录被临时禁止了。在维护归档目录时，会经常将目录设置为这个状态。
- ALTERNATIVE：说明该目录是备用目录。当与其对应的主目录发生故障时，会自动启动备用目录。

在讨论归档目录状态后，我们重点分析 DEFER 和 ALTERNATIVE。首先我们设置归档目录 log_archive_dest_1 的 DEFER 状态，在设置之前我们查看该归档目录的状态。

例子 12-11　查看该归档目录的状态

```
    SQL> select dest_name,status,binding,process ,error from v$archive_dest where
status NOT like 'INA%';

    DEST NAME           STATUS    BINDING          PROCESS    ERROR
    -------------------------------------------------------------------------
    LOG_ARCHIVE_DEST_1   VALID     MANDATORY        ARCH
    LOG ARCHIVE DEST 2   VALID     OPTIONAL         ARCH
    LOG_ARCHIVE_DEST_3   VALID     OPTIONAL         ARCH
```

从输出知道归档目录 log_archive_dest_1 的当前状态为 VALID。下面修改该目录的状态为 DEFER，临时禁止该归档目录的使用。

例子 12-12　修改归档目录的状态为 DEFER

```
    SQL> alter system set log_archive_dest_state_1=defer;

    系统已更改。
```

此时，如果发生归档行为，则归档日志无法使用该目录，虽然该目录为 mandatory，在笔者手工归档时，依然可以归档成功，因为我们设置了 log_archive_min_succeed_dest 参数为 2，并且设置了一个 mandatory 归档目录和两个 optional 归档目录。

例子 12-13　查询归档目录 log_archive_dest_1 的状态

```
    SQL> select dest name,status,binding,process ,error from v$archive dest where
status NOT like 'INA%';

    DEST_NAME                STATUS    BINDING       PROCESS    ERROR
```

```
-------------------------------------------------------------------------
LOG_ARCHIVE_DEST_1  DEFERRED  MANDATORY  ARCH
LOG_ARCHIVE_DEST_2  VALID      OPTIONAL    ARCH
LOG_ARCHIVE_DEST_3  VALID      OPTIONAL    ARCH
```

此时归档目录 log_archive_dest_1 的状态已经发生变化，此时归档进程将无法使用该目录，在维护期间发生的归档日志必须手工复制到该目录下，以防止归档日志的缺失。

上面是在目录没有发生故障时我们做了一次状态转变，下面我们恢复该目录的状态，然后制造一个故障深化对 DEFER 状态的学习。

例子 12-14　恢复目录 log_archive_dest_1 的状态为 VALID

```
SQL> alter system set log_archive_Dest_state_1=enable;

系统已更改。
```

上述指令执行成功后，继续查询 v$archive_dest 视图，此时目录 log_archive_dest_1 的状态将恢复为 valid。读者可以自行查询。下面我们修改参数 log_archive_dest_1 的目录为 location=f:\app\archive 来模拟该目录损坏，然后多次切换日志。

```
SQL> alter system switch logfile;

系统已更改。

SQL> /

系统已更改。

SQL> /
```

上述我们连续 3 次切换了重做日志，此时光标会在第 3 次切换处停顿，无法切换成功。我们分析原因，因为我们有 3 个重做日志组，经过日志切换后，最初那个重做日志组需要被覆盖使用，但是由于设置为 mandatory 属性的归档目录损坏，所以最初那个重做日志没有归档到 mandatory 属性的归档目录下，而这是不允许的。也就是 mandatory 属性的归档目录必须归档成功，否则该归档日志无法被覆盖使用。其实从 mandatory（强制）的含义也可以理解。观察 alert 日志会发现如下记录。

```
hread 1 advanced to log sequence 125 (LGWR switch)
  Current log# 2 seq# 125 mem# 0: F:\APP\ORACLE\ORADATA\ORCL\REDO02.LOG
Fri Feb 01 09:50:30 2013
Errors in file f:\app\oracle\diag\rdbms\orcl\orcl\trace\orcl_arc3_2800.trc:
ORA-09291: sksachk: invalid device specified for archive destination
OSD-04018: 无法访问指定的目录或设备。
O/S-Error: (OS 2) 系统找不到指定的文件。
Cannot translate archive destination string 'LOG_ARCHIVE_DEST_1'
Archived Log entry 46 added for thread 1 sequence 124 ID 0x4f337ebc dest 2:
Archived Log entry 47 added for thread 1 sequence 124 ID 0x4f337ebc dest 3:
ARCH: Archival stopped, error occurred. Will continue retrying
Errors in file f:\app\oracle\diag\rdbms\orcl\orcl\trace\orcl arc3 2800.trc:
ORA-16038: log 1 sequence# 124 cannot be archived
ORA-09291: sksachk: invalid device specified for archive destination
ORA-00312: online log 1 thread 1: 'F:\APP\ORACLE\ORADATA\ORCL\REDO01.LOG'
```

```
    Thread 1 cannot allocate new log, sequence 126
```

我们重新打开一个连接继续查询 v$archive_dest 视图，如例子 12-15 所示。

例子 12-15　查询归档终点状态

```
    SQL>select dest_name,status,binding,process ,error from v$archive_dest where
status NOT like 'INA%'

    DEST_NAME            STATUS    BINDING   PROCESS    ERROR
    ------------------------------ ----------------------------------------
    LOG_ARCHIVE_DEST_1    ERROR     MANDATORY ARCH    ORA-16032: 无法转换参数
                                                          的目标字符串
    LOG_ARCHIVE_DEST_2    VALID     OPTIONAL  ARCH
    LOG_ARCHIVE_DEST_3    VALID     OPTIONAL  ARCH
```

从输出可知 LOG_ARCHIVE_DEST_1 的状态为 ERROR，错误信息为无法转换字符串，说明该目录不存在（这是我们制造的错误）。为了处理这个问题，我们需要使用 DEFER 状态。下面我们修改参数 LOG_ARCHIVE_DEST_STATE_1 的值。

```
    SQL> alter system set log_archive_dest_state_1=defer;

    系统已更改。
```

下面先验证修改结果。

```
    SQL> select dest_name,status,binding,process ,error from v$archive_dest where
status NOT like 'INA%';
    DEST_NAME             STATUS     BINDING      PROCESS    ERROR
    --------------------------------------------------------------------------
    LOG_ARCHIVE_DEST_1    DISABLED  MANDATORY  ARCH     ORA-16032: 无法转换
                                                            参数  的目标字符串
    LOG_ARCHIVE_DEST_2    VALID     OPTIONAL    ARCH
    LOG_ARCHIVE_DEST_3    VALID     OPTIONAL    ARCH
```

此时 LOG_ARCHIVE_DEST_1 的状态为 DISABLED 说明该目录无法使用。我们修复该目录为 location=f:\app\archive1\并重新使用，将该目录的状态修改为 ENABLE。

```
    SQL> alter system set log_archive_dest_state_1=enable;

    系统已更改。
```

此时，另一个窗口的归档会继续执行，说明该归档目录可用，我再次通过查询验证修复结果。

例子 12-16　验证归档目录修复结果

```
    SQL> select dest_name,status,binding,process ,error from v$archive_dest where
status NOT like 'INA%';

    DEST_NAME             STATUS    BINDING      PROCESS    ERROR
    --------------------------------------------------------------------------
    LOG_ARCHIVE_DEST_1    VALID     MANDATORY  ARCH
    LOG_ARCHIVE_DEST_2    VALID     OPTIONAL     ARCH
    LOG_ARCHIVE_DEST_3    VALID     OPTIONAL     ARCH
```

接着我们讨论 ALTERNATIVE，首先对 LOG_ARCHIVE_DEST_1 设置一个备用的可选目录，这个目录的作用就是一旦当前目录损坏，备用目录可以继续使用，不会发生数据库 HANG 住的情况。下面是设置步骤。

01 设置一个归档终点目录，此时设置参数 log_archive_dest_4。

```
SQL> alter system set log_archive_dest_4='location=f:\app\archive4\';

系统已更改。
```

02 修改参数 log_archive_dest_1 的设置，增加 alternate 参数，该参数使用 LOG_ARCHIVE_DEST_4 作为备用归档目录。

```
SQL> alter system set log archive dest 1='location=f:\app\archive1\ mandatory
     alternate=LOG_ARCHIVE_DEST_4';

系统已更改。
```

03 将 LOG_ARCHIVE_DEST_4 归档目录的状态修改为 alternate。

```
SQL> alter system set log archive dest state 4=alternate;

系统已更改。
```

在完成上述 3 个步骤后，通过下面的查询验证我们的修改结果。首先查看归档目录的修改，我们为归档目录 log_archive_Dest_1 增加了一个备用目录，如下例所示。

例子 12-17　验证目录 log_archive_Dest_1 的修改结果

```
SQL> show parameter log_archive_Dest_1

NAME                    TYPE        VALUE
------------------------------------------------------------------------------
log_archive_dest_1      string      location=f:\app\archive1\ mand
                        atory       alternate=LOG_ARCHIVE_DEST_4
```

最后通过 v$archive_dest 视图查询所有设置的归档终点名称和状态，如下例所示。

例子 12-18　查询所有的归档终点名称和状态

```
SQL> SELECT dest_name, status, destination FROM v$archive_dest where status
NOT like 'INA%'

DEST NAME               STATUS      DESTINATION
-------------------- --------- ----------------------------------------
LOG_ARCHIVE_DEST_1      VALID       f:\app\archive1\
LOG ARCHIVE DEST 2      VALID       f:\app\archive2
LOG ARCHIVE DEST 3      VALID       f:\app\archive3\
LOG_ARCHIVE_DEST_4      ALTERNATE   f:\app\archive4\
```

当参数 LOG_ARCHIVE_DEST_1 指定的目录正常时，ALTERNATE 属性的目录不会被使用，如果发生归档，ALTERNATE 属性目录下不会出现任何归档。

12.5　本章小结

本章我们学习了 Oracle 归档管理，对于 DBA 归档是需要认真维护的文件。一旦数据库处于归档模式， OLTP 系统会产生大量的归档日志，需要认真调整归档进程以及设置归档目录，以保证归档的效率和归档的高可用性。这些内容本章都做了详细介绍，最后对于归档目录的维护介绍了 DEFER 状态和 ALTERNATE 状态。

第 13 章

表空间与数据文件管理

为了管理数据文件，Oracle 提出了表空间的概念，Oracle 将数据逻辑存储在表空间中，但实际上是存储在数据文件中。本章将讲解 Oracle 引入的逻辑机构和物理结构、二者之间的关系，以及 DBA 需要熟练操纵的表空间管理。读者通过本章的学习，应该掌握如何创建表空间（数据字典管理和本地管理）、如何设置表空间为脱机或只读状态，以及如何修改表空间中数据文件的大小等。

13.1 Oracle数据库的逻辑结构

Oracle 曾经声称，自己的数据库系统具有跨平台特性，在一个数据库平台上开发的数据库可以不加修改地移植到另一个操作系统平台上。这样 Oracle 就不会直接操作底层操作系统的数据文件，而是提供一个中间层，这个中间层就是 Oralce 的逻辑结构，它与操作系统的平台无关，而中间层到数据文件的映射通过 DBMS 来完成，如图 13-1 所示。

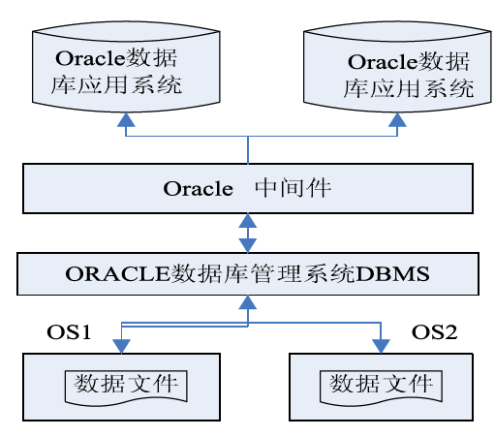


图 13-1 Oracle 的跨平台特性图

如图 13-1 所示，Oracle 数据库应用系统通过操作中间件来实现逻辑操作，而逻辑操作到数据文件操作之间是通过 DBMS 来映射完成的。这样数据文件对于 Oracle 数据库应用系统是透明的。

我们再给出 Oracle 为了管理数据文件而引入的逻辑结构，该逻辑结构也正是图 13-1 中的中间件的内容，如图 13-2 所示。该图展示了数据文件管理的逻辑结构和物理结构。图的左边是逻辑结

构。逻辑结构从上到下是包含关系，也是一对多的关系，即一个数据库有一个或多个表空间，一个表空间有一个或多个段，而一个段由一个或多个区段组成，一个区段由多个数据库块组成，一个数据库块由多个操作系统数据库块组成。

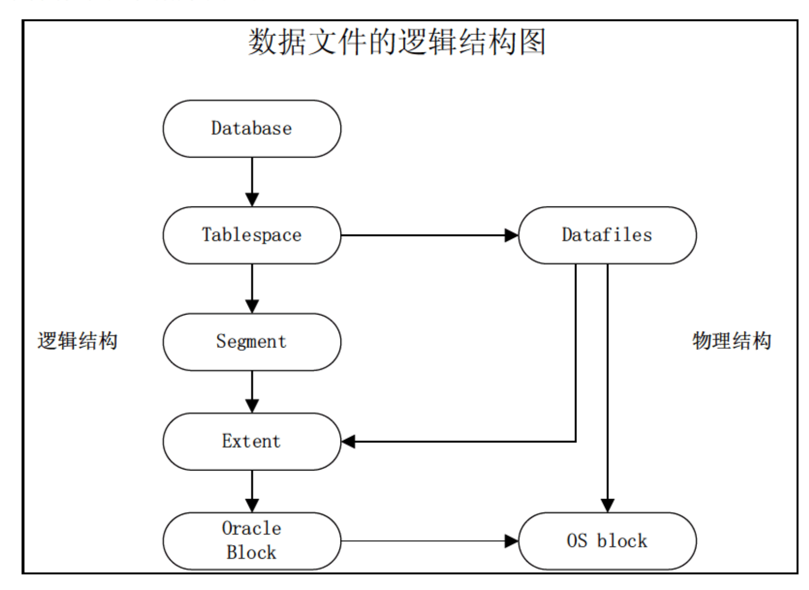


图 13-2　数据文件管理的逻辑结构图

右边是物理结构，一个表空间有一个或多个数据文件，一个数据文件物理上由操作系统块组成。下面详细解释逻辑结构的各种组成。

- 表空间（Tablespace）：在逻辑上一个数据库由表空间组成，一个表空间只能属于一个数据库，而反之不成立，一个表空间包括一个或多个操作系统文件，把这些操作系统文件称为数据文件。表空间包含一个或多个段。
- 段（Segment）：段是表空间内的一个逻辑存储空间，一个表空间包含一个或多个段，一个段不能跨表空间（即一个段只能在一个表空间中），但是段可以跨越数据文件，即一个段可以分布在同一个表空间的几个数据文件上。一个段由一个或多个区段组成。
- 区段（Extent）：区段是段中分配的逻辑存储空间，一个区段由连续的 Oracle 数据库块组成，一个区段只能存在于一个数据文件中。但创建一个段时，该段至少包含一个区段，但段增长时，将分配更多的区段给该段，同时 DBA 可以手动地向段中添加区段。
- 数据库块（Oracle block）：Oracle 数据库服务器管理存储空间中的数据文件的最小单位就是数据库块，它是 Oracle 数据库系统输入输出的最小单位。一个数据库块由一个或多个操作系统块组成。Oracle 提供了标准的数据库块尺寸，该尺寸通过初始化参数 DB_BLOCK_SIZE 设置，在初始创建数据库时指定。一个数据库块应该是操作系统块的整数倍，这样可以避免不必要的 I/O。数据库块在创建数据库时由初始化参数 DB_BLOCK_SIZE 设置，一般大小为 4KB 或 8KB。可以使用例子 13-1 来查看各个表空间所使用的数据库块的大小。

例子 13-1　查看表空间的数据库块大小

```
SQL> select tablespace name,block size,contents
  2  from dba_tablespaces;

TABLESPACE NAME                BLOCK SIZE CONTENTS
------------------------------------------------------------------------------
SYSTEM                              4096     PERMANENT
UNDOTBS                             4096     UNDO
CWMLITE                             4096     PERMANENT
DRSYS                               4096     PERMANENT
EXAMPLE                             4096     PERMANENT
INDX                                4096     PERMANENT
TEMP                                4096     TEMPORARY
TOOLS                               4096     PERMANENT
USERS                               4096     PERMANENT
RBS                                 4096     PERMANENT

已选择 10 行。
```

例子 13-1 的输出结果说明该数据库中的表空间的数据库块尺寸为 4KB（4096B）。

数据文件物理结构的各组成部分如下。

- 数据文件（datafile）：数据文件是 Oracle 格式的操作系统文件，即.dbf 文件等。数据文件的大小决定了表空间的大小，当表空间不足时，需要增加新的数据文件或者重新设置当前数据文件的大小，以满足表空间增长的需求。
- 操作系统块（OS block）：操作系统数据块依赖于不同的操作系统平台，它是操作系统操作数据文件的最小单位，一个或多个操作系统块组成了一个数据库块。

13.2　表空间的分类以及创建表空间

表空间是 Oracle 数据库的逻辑结构，数据逻辑地存储在表空间中，而实际上是存储在物理的操作系统文件中，该文件是 Oracle 格式，一个表空间由一个或多个数据文件组成，数据文件不能跨表空间存储，即一个数据文件只能属于一个表空间。

在一个数据库中表空间的数量没有严格限制，大小为 2GB 的表空间和大小为 20MB 的表空间可以并存，只有用户根据业务需求赋予表空间的功能不同（系统表空间除外）。有几个表空间是所有 ORACLE 数据库必备的表空间，它们是 System 表空间、临时表空间、还原表空间和默认表空间。在 Oracle 11g 中还有 Sysaux 表空间，该表空间是 System 表空间的扩充，包含各种 Oracle 产品和功能部件使用的数据。虽然还原表空间、默认表空间以及临时表空间可以使用 System 表空间，但是这些表空间是必须的。

Oracle 数据库把表空间分为两类：系统表空间和非系统表空间。

系统表空间顾名思义是数据库系统创建时需要的表空间，这些表空间在数据库创建时自动创建，是每个数据库必须的表空间，满足数据库系统运行的最低要求，如系统表空间（SYSTEM）中存放数据字典或者存放还原段。在用户没有创建非系统表空间时，系统表空间可以存放用户数据或

索引，但是这样做会增加系统表空间的 I/O，影响系统效率。

非系统表空间是用户根据业务需求而创建的表空间，非系统表空间可以按照数据多少、使用频度、需求数量等方面灵活设置，这些表空间可以存储还原段或临时段，即创建还原表空间和临时表空间（默认是在系统表空间中），这样一个表空间的功能就相对独立，在特定的数据库应用环境下可以很好地提高系统的效率。通过创建用户自定义的表空间，如还原表空间、临时表空间、数据表空间或者索引表空间，使得数据库的管理更加灵活、方便。

创建表空间的语法为：

```
CREATE TABLESPACE tablespac ename
[DATAFILE clause ]
[MINIMUM EXTENT integer[k|m]]
[BLOCKSIZE integer[k]]
[LOGGING|NOLOGGING]
[DEFAULT storage clause]
[ONLINE|OFFLINE]
[PERMANENT|TEMPORARY]
```

下面依次解释这些子句的含义。

- DATAFILE 子句：组成该表空间的数据文件的名字，该子句应该给出完整的目录和文件名，目录必须存在否则无法创建成功。
- MINIMUM EXTENT：定义该表空间中最小的区段的大小，这样该表空间中的区段大小为该最小值的整数倍。
- BLOCKSIZE：指出该表空间使用的非标准块尺寸的大小，单位为 K。在设置该参数前必须相应的先修改参数文件中的两个参数即 DB_CACHE_SIZE 和 DB_nK_CACHE_SIZE，并且这两个参数的值必须与 BLOCKSIZE 的值相同。但使用默认标准块尺寸时，参数 DB_nK_CACHE_SIZE 的值为 0。
- [LOGGING|NOLOGGING]：该参数说明是否把该表空间中数据的变化记录在重做日志文件中。LOGGING 为记录变化，NOLOGGING 为不记录变化。
- DEFAULT 子句：该子句定义该表空间的一些参数，如初始区段的尺寸、最小区段尺寸、最大区段数量等。
- [ONLINE|OFFLINE]：该参数指出该表空间创建后是否联机，ONLINE 为创建后立即联机，OFFLINE 为创建后不联机。默认为联机状态。
- [PERMANENT|TEMPORARY]：参数 PERMANENT 表示该表空间只能存储永久对象，TEMPORARY 说明该表为临时表空间，该表空间只能用户临时存储非永久对象，如用户使用 ORDER BY 子句查询数据时的排序结果，临时表空间不需要将变化记录到重做日志文件，它只包含用户会话期间的数据，如排序中间结果等。不使用该参数则默认为永久表空间。

下面通过例子 13-2 说明如何创建表空间。我们创建一个表空间，表空间名为 user_data，该表空间用来存储用户表，表空间就包含一个数据文件，大小为 100MB，文件名为 d:\userdata\userdata1.dbf。

例子 13-2　创建表空间 user_data

```
SQL> create tablespace user data
  2  datafile 'd:\userdata\userdata1.dbf' size 100 M
表空间已创建。
```

此时查看该表是否创建，如例子 13-3 所示。

例子 13-3　查看该表空间 user_data 是否创建

```
SQL> select tablespace_name,logging,stat
  2  from dba tablespaces;

TABLESPACE NAME      LOGGING   STATUS
-------------------------------------- ----------------- -----------
SYSTEM                LOGGING    ONLINE
UNDOTBS               LOGGING    ONLINE
CWMLITE               LOGGING    ONLINE
DRSYS                 LOGGING    ONLINE
EXAMPLE               LOGGING    ONLINE
INDX                  LOGGING    ONLINE
TEMP                  NOLOGGING ONLINE
TOOLS                 LOGGING    ONLINE
USERS                 LOGGING    ONLINE
RBS                   LOGGING    ONLINE
USER DATA             LOGGING    ONLINE

已选择 11 行。
```

表空间 user_data 创建成功，且处于记录日志模式，该表空间目前处于在线状。接着，我们查看该表空间对应的数据文件是否存在，如例子 13-4 所示。

例子 13-4　查看表空间 user_data 对应的数据文件

```
SQL> select file_name,tablespace_name,status
  2  from dba data files
  3  where tablespace name = 'USER DATA';

FILE_NAME                  TABLESPACE_NAME      STATUS
-------------------------------------------------------------------------
D:\USERDATA\USERDATA1.DBF   USER_DATA           AVAILABLE
```

显然，表空间 user_data 中包含一个数据文件，该数据文件位于目录 D:\USERDATA 下，文件名为 USERDATA1.DBF。

这里没有演示创建表空间的全部参数，因为在有些条件下某些参数是不能创建的，在下节讲完表空间的两种管理方案后，我们再具体学习何种情况下，以及如何使用这些参数创建满足用户需求的表空间。

13.3　表空间磁盘管理的两种方案

Oracle 提供了两种管理表空间的区段的方案，一种是数据字典管理，一种是本地管理，其实管理表空间的实质就是为用户分配可用的区段以及回收空闲区段的过程。两种管理表空间的方式体现在对表空间区段的管理方式不同，系统效率的影响也有所不同。Oracle 推荐使用本地管理表空间的方式，在 Oracle 9i 及其以上版本中默认创建的表空间是本地管理的。

13.3.1　数据字典管理的表空间磁盘管理

我们已经讲过表空间的管理就是对该表空间区段的管理，但用户插入数据或表中的其他对象（如索引等）增加时，就需要分配更多的区段给新增的数据使用。

数据字典管理的方式是将每个数据字典管理的表空间的使用情况记录在数据字典的表中，当分配或撤销表空间区段的分配时，则隐含地使用 SQL 语句对表操作以记录当前表空间区段的使用情况，并且在还原段中记录了变换前的区段使用情况，就像操作普通表时 Oracle 的行为一样，显然这种方式增加了数据字典的频繁操作，对于一个大型的数据库系统，有几百个甚至上千个表空间需要管理，可想象这样的系统效率会如何。

Oracle 已经注意到数据字典管理表空间的问题，所以引入了本地管理的表空间。

13.3.2　本地管理的表空间磁盘管理

本地管理的表空间是为了解决数据字典管理表空间效率不高的问题，Oracle 设计让每一个表空间自己管理表空间区段的分配，记录区段的使用情况。

在数据文件头中有一个区域用于存储本地管理的表空间的数据文件的空间信息，将表空间中数据文件的可用和已用空间信息记录下来。显然这样的管理方式类似于一种分布式管理，减轻了数据字典的工作压力。

本地管理的方式使用位图在数据文件头中记录数据文件的可用和已用信息，位图使用一个数据位表示一个数据库块或者一组数据库块的使用情况，而表空间分配的最小单位为区段 EXTENT，而一个区段由多个数据库块组成，所以当需要在表空间中增加新对象时，就需要查找位图，看是否有一段连续的 Oralce 数据库块空闲。

显然，不使用数据字典管理的表空间提高了系统效率，解决了数据字典的瓶颈问题，但是任何事物都有两面，使用本地管理的表空间不能随意更改默认的存储参数，如初始区段的大小，最大区段数等，但是效率的提高足以弥补本地管理表空间的不足。所以，Oracle 极力推荐使用本地管理的表空间。

在下面一节将详细讲述如何创建数据字典管理的表空间和本地管理的表空间。

13.4　创建表空间

在一个生产数据库中，往往存在大量的表空间，根据业务需要将用户表或其他对象保存在表

空间中，从而根据硬件环境来减少数据库的 I/O，也方便数据空间的维护。本节我们通过实例演示如何创建数据字典管理的表空间和本地管理的表空间。在 Oracle 9i 及以上版本中本地管理的表空间为默认设置，它和创建数据字典管理的表空间有些不同，我们先介绍如何创建数据字典管理的表空间。

13.4.1　创建数据字典管理的表空间

我们创建一个数据字典管理的表空间，该表空间有 3 个数据文件，分别放在不同的磁盘以及目录下，这样做的目的是有利于平衡 I/O，每个数据文件的大小为 100MB，最小区段为 20KB，默认存储参数为：初始区段大小为 20KB，当再次分配区段（EXTENT）时，分配的区段大小也为 20KB，所分配的最大磁盘空间为 500 个区段，如例子 13-5 所示。

例子 13-5　创建数据字典管理的表空间

```
SQL> create tablespace tianjin data
  2  datafile 'd:\userdata\tianjin01.dbf' size 100M,
  3        'e:\userdata\tianjin02.dbf' size 100M,
  4        'f:\userdata\tianjin03.dbf' size 100M
  5  minimum extent 20k
  6  extent management dictionary
  7  default storage(initial 20k next 20k maxextents 500 pctincrease 0);

表空间已创建。
```

此时，我们成功创建了一个表空间，该表空间有 3 个数据文件分别存放在不同的磁盘上，我们通过例子 13-6 验证创建结果。

例子 13-6　验证例子 13-5 中创建的表空间

```
SQL> select tablespace name,extent management
  2  from dba_tablespaces
  3  where tablespace_name ='TIANJIN_DATA ';

TABLESPACE NAME              EXTENT MAN
---------------------------------------------------- --------------------
TIANJIN_DATA                 DICTIONARY
```

此时在数据字典 DBA_TABLESPACES 中可以查询到例子 13-5 创建的数据字典，并且其区段管理方式为 DICTIONARY。

在表空间 TIANJIN_DATA 中，我们创建了 3 个数据文件，分别存放在不同的磁盘上，下面通过例子 13-7 验证这些数据文件的信息。

例子 13-7　验证表空间 TIANJIN_DATA 中的数据文件

```
SQL> col file name for a30
SQL> col tablespace name for a20
SQL> run
  1  select tablespace name,file name,blocks,status
  2  from dba data files
  3* where tablespace_name = 'TIANJIN_DATA'
```

```
TABLESPACE_NAME     FILE_NAME                   BLOCKS STATUS
------------------------------------------- ----------- ----------------
TIANJIN DATA     D:\USERDATA\TIANJIN01.DBF     25600 AVAILABLE
TIANJIN DATA     E:\USERDATA\TIANJIN02.DBF     25600 AVAILABLE
TIANJIN_DATA     F:\USERDATA\TIANJIN03.DBF     25600 AVAILABLE
```

例子 13-7 的输出说明，数据字典 TIANJIN_DATA 有 3 个数据文件，且数据文件的状态都为 AVAILABLE。

在例子 13-5 中我们使用了一些默认存储参数，我们再用例子 13-8 验证这些参数。

例子 13-8　查询表空间 TIANJIN_DATA 的默认存储参数

```
    SQL> set line 100
    SQL>select
tablespace_name,block_size,initial_extent,next_extent,max_extents,pct_increase
      2  from dba tablespaces
      3* where tablespace name = 'TIANJIN DATA'
    TABLESPACE_NAME          BLOCK_SIZE   INITIAL_EXTENT   NEXT_EXTEN   MAX_EXTENTS
PCT_INCREASE
    ------------------------------------------------------------- ------------
    TIANJIN_DATA                4096           20480        20480          500            0
```

例子 13-8 的输出说明，表空间 TIANJIN_DATA 的第一次分配区段时，区段的大小为 20 480B（即 20KB），再次分配区段时，区段大小也为 20 480B，最大区段数量为 500 个区段。

13.4.2　创建本地管理的表空间

我们创建一个本地管理的表空间，该表空间名字为 BEIJING_DATA，由于本地管理的表空间不能随意地更改存储参数，所以创建起来较之创建数据字典管理的表空间要简洁一些。如例子 13-9 所示，创建本地管理的表空间。

例子 13-9　创建本地管理的表空间

```
SQL> create tablespace beijing_data
  2  datafile 'd:\userdata\beijingdata01.dbf' size 100M
  3  extent management local
  4  uniform size 1M;

表空间已创建。
```

在该例子中，创建了名为 BEIJING_DATA 的表空间，该表空间只有一个数据文件（大小为 100MB），区段管理方式为本地管理，区段尺寸同一为 1MB。下面通过例子 13-10 验证表空间 BEIJING_DATA 的区段管理方式。

例子 13-10　验证表空间 BEIJING_DATA 的区段管理方式

```
SQL> select tablespace name,block size,extent management,status
  2  from dba tablespaces
  3* where tablespace name like 'BEIJING%'
```

```
TABLESPACE NAME      BLOCK SIZE EXTENT MAN STATUS
-------------------- ---------- ---------- ---------
BEIJING_DATA               4096 LOCAL      ONLINE
```

例子 13-10 的输出结果表明，表空间 BEIJING_DATA 为本地管理，因为其 EXTENT_MAN 为 LOCAL，且默认该表空间一旦创建就是联机状态，因为 STATUS 为 ONLINE。

例子 13-11　验证表空间 BEIJING_DATA 的数据文件信息

```
SQL> select tablespace_name,file_name, status
  2  from dba data files
  3  where tablespace name ='BEIJING DATA';

TABLESPACE_NAME      FILE_NAME                                STATUS
-----------------------------------------------------------------------
BEIJING_DATA         D:\USERDATA\BEIJING_DATA01.DBF           AVAILABLE
```

输出说明新建的表空间 BEIJING_DATA 中只有一个数据文件，该文件存储在目录 D:\USERDATA 下，文件名为 BEIJING_DATA01.DBF。

在建立本地管理的表空间时，我们没有使用默认存储参数，只是使用了一个 UNIFORM SIZE 参数，设置统一的区段尺寸，下面通过例子 13-12 来验证该本地管理的表空间的存储参数信息。

例子 13-12　查看本地管理的表空间的存储参数信息

```
SQL>                                                                    select
tablespace_name,block_size,initial_extent,next_extent,max_extents,pct_increase
  2  from dba tablespaces
  3  where tablespace name = 'BEIJING DATA';

TABLESPACE_NAME     BLOCK_SIZE    INITIAL_EXTENT     NEXT_EXTENT     MAX_EXTENTS
PCT INCREASE
------------------------- ---------- ------------- ----------- ------------
BEIJING_DATA    4096        1048576         1048576        2147483645            0
```

表空间 BEIJING_DATA 的初始区段大小为 1MB，再次分配区段时区段大小也为 1MB，数据库块尺寸为默认标准块尺寸 4KB。

13.4.3　创建还原表空间

还原表空间存放还原段。这里通过例子说明还原段的作用，如果一个用户要修改某个属性值，把月薪金额从 2000 改到 2500，在更改的过程中其他用户要查看该数据时，看到的应该是 2000，因为当前用户正在更改数据，还没有提交。所以为了保证这种读一致性，Oracle 设计了还原段，在还原段中存放更改前的数据。

还原表空间只能存放还原段，不能存放其他任何对象。在创建还原表空间时，只能使用 DATAFILE 子句和 EXTENT MANAGEMENT 子句。通过例子 13-13 演示如何创建还原表空间。

例子 13-13　创建还原表空间 USER_UNDO

```
SQL> create undo tablespace user undo
  2  datafile 'd:\userundo\user undo.dbf'
  3* size 30M
```

```
表空间已创建。
```

同样，我们通过例子 13-14 验证是否成功创建还原表空间 USER_UNDO。

例子 13-14　查看是否成功创建还原表空间 USER_UNDO

```
SQL> select tablespace name,status,contents,logging,extent manageme
  2  from dba_tablespaces;

TABLESPACE NAME  STATUS    CONTENTS  LOGGING   EXTENT MAN
--------------------------------- ----------------- --------------------
SYSTEM                  ONLINE    PERMANENT LOGGING   DICTIONARY
UNDOTBS                ONLINE    UNDO      LOGGING   LOCAL
CWMLITE                ONLINE    PERMANENT LOGGING   LOCAL
DRSYS                   ONLINE    PERMANENT LOGGING   LOCAL
EXAMPLE                ONLINE    PERMANENT LOGGING   LOCAL
INDX                     ONLINE    PERMANENT LOGGING   LOCAL
TEMP                    ONLINE    TEMPORARY NOLOGGING LOCAL
TOOLS                   ONLINE    PERMANENT LOGGING   LOCAL
USERS                   ONLINE    PERMANENT LOGGING   LOCAL
RBS                     ONLINE    PERMANENT LOGGING   LOCAL
USER_UNDO               ONLINE    UNDO      LOGGING   LOCAL

已选择 11 行。
```

在上述输出的最后一行可以看出表空间 USER_UNDO 已经创建，该表空间的状态为联机状态，CONTENTS 为 UNDO 说明它是还原表空间，LOGGING 说明该表空间的变化受重做日志的保护，区段的管理方式为本地管理。

下面我们通过例子 13-15 验证新创建的还原表空间的存储参数设置信息。

例子 13-15　查看还原表空间 USER_UNDO 的存储参数

```
SQL> select tablespace_name,block_size,initial_extent,next_extent,max_extents
  2  from dba tablespaces
  3  where contents = 'UNDO';

TABLESPACE_NAME  BLOCK_SIZE INITIAL_EXTENT NEXT_EXTENT MAX_EXTENTS
-------------------------------- ------------------- ----------------------
UNDOTBS                    4096         65536          2147483645
USER_UNDO                  4096         65536          2147483645
```

我们查询了内容为 UNDO 的表空间信息，查询结果说明当前的数据库有两个还原表空间，其中 UNDOTBS 是系统创建的。在 Oracle 11g 中，虽然只有 SYSTEM 和 SYSAUX 表空间是强制创建的，但是在安装时会默认创建一个 UNDOTBS1 的还原表空间，创建一个还原数据文件，默认文件名为 UNDOTBS01.DBF。在例子 13-13 中表空间 USER_UNDO 是刚刚创建的，它的默认数据库块尺寸为 4096 字节，初始区段大小为 65 536 字节，而可分配的最大区段不是一个数量值，和数据字典管理的表空间不同，它是一个数据位数，因为本地管理的表空间是通过数据位记录空间的数据库块的使用情况。

在创建还原表空间时，创建了一个数据文件，我们通过例子 13-16 查看还原表空间中数据文件

的信息。

例子 13-16　查看还原表空间 USER_UNDO 的数据文件

```
SQL> select file name,file id,tablespace name,status
  2  from dba_data_files
  3* where tablespace_name = 'USER_UNDO'

FILE NAME                    FILE ID     TABLESPACE NAME     STATUS
---------------------------------------------------------    --------------
D:\USERUNDO\USER_UNDO.DBF   9           USER_UNDO           AVAILABLE
```

还原表空间 USER_UNDO 中的数据文件为 D:\USERUNDO\USER_UNDO.DBF，该文件当前可以使用，因为 STATUS 为 AVAILABLE。

在用户创建了还原表空间后如果需要可以把当前数据库正在使用的还原表空间切换到新建立的还原表空间上。

13.4.4　创建临时表空间

在 Oracle 数据库中临时表空间用于用户的特定会话活动，如用户会话中的排序操作，排序的中间结果需要存储在某个区域，这个区域就是临时表空间，临时表空间的排序段是在实例启动后有第一个排序操作时创建的。如果在创建数据库时没有创建临时表空间则数据库服务器默认使用 SYSTEM 表空间，显然这样会影响数据库系统的效率，因为 SYSTEM 表空间中存储了数据字典等数据库系统的一些重要信息。在 Oracle 9i 中会自动创建一个 TEMP 的临时表空间，在 Oracle 11g 中，只用 SYSTEM 和 SYSAUS 表空间是强制建立的，临时表空间会默认创建，名字为 TEMP，在该表空间中会创建一个临时数据文件，文件名为 TEMP01.DBF，它和其他数据文件放在同一个目录下，该目录为$ORACLE_HOME/ORADATA/ORALCE_ID。

临时表空间是使用当前数据库的多个用户共享使用的，临时表空间中的区段在需要时按照创建临时表空间时的参数或管理方式进行扩展。

下面演示在数据库创建完成后，如何创建一个临时表空间，如例子 13-17 所示。

例子 13-17　创建一个临时表空间 USER_TEMP

```
SQL> create temporary tablespace user temp
  2  tempfile 'd:\usertemp\user_temp.dbf' size 20MB
  3  extent management local
  4  uniform size 1M;

表空间已创建。
```

在创建临时表空间时，需要使用 CREATE TEMPORARY 告诉数据库服务器该表空间是临时表空间，并且表空间中的数据文件必须使用 TEMPFILE 标识它是临时表空间的数据文件。

在例子 13-17 中，创建的临时表空间名为 USER_TEMP，区段管理方式为本地管理，区段的统一扩展尺寸是 1MB。

下面通过例子 13-18 查询是否成功创建临时表空间 USER_TEMP。

例子 13-18　查询是否成功创建临时表空间 USER_TEMP

```
SQL> select tablespace name,status,contents,logging
  2  from dba_tablespaces
  3  where tablespace_name like 'USER%';

TABLESPACE NAME            STATUS     CONTENTS    LOGGING
-------------------------- ---------- ----------- ----------
USERS                      ONLINE     PERMANENT   LOGGING
USER TEMP                  ONLINE     TEMPORARY   NOLOGGING
USER_UNDO                  ONLINE     UNDO        LOGGING
```

表空间 USER_TEMP 为临时表空间，因为 CONTENTS 为 TEMPORARY，它处于联机状态，尤其需要注意的是该表空间为 NOLOGGING，即不需要将临时表空间的变化记录到重做日志文件中。

在例子 13-17 中，我们在临时表空间中创建了一个数据文件，该文件 D:\USER_TEMP\USER_TEMP.DBF，大小为 20MB，我们通过数据字典视图 v$tempfile 来查看该数据文件信息，如例子 13-19 所示。

例子 13-19　通过数据字典视图 v$tempfile 来查看数据文件信息

```
SQL> col name for a30
SQL> select file#,status,enabled,bytes,block size,name
  2  from v$tempfile;

FILE#  STATUS  ENABLED     BYTES     BLOCK   SIZE NAME
-----------------------------------------------------------------------
1      ONLINE  READ WRITE  20971320  4096    D:\USERTEMP\USER_TEMP.DBF
```

通过输出结果说明，该临时数据文件为可读可写的，当前处于联机状态，文件大小为 20MB，数据块尺寸为标准尺寸 4KB。

临时表空间中的临时数据文件也是.DBF 格式的数据库格式文件，但是这个数据文件和普通的存储表或索引的数据文件有所不同。

- 临时文件总是处于 NOLOGGING 模式，因为临时表空间中的数据都是中间数据，只是临时存放的，它们的变化不需要记录到重做日志文件中，因为这些变化本身也不需要恢复。
- 临时文件不能设置为只读（read_only）模式.
- 临时文件不能重命名。
- 临时文件不能通过 ALTER DATABASE 创建。
- 临时文件用于只读数据库。
- 介质恢复时不需要临时文件。
- 使用 BACKUP CONTROLFILE 并不产生任何关于临时文件的信息。
- 使用 CREATE CONTROLFILE 不能设置任何与临时文件有关的信息。

在初始化参数文件中，有一个参数为 SORT_AREA_SIZE，这是排序区尺寸大小，为了优化临时表空间中排序操作的性能，最好将 UNIFORM SIZE 设置为 SORT_AREA_SIZE 的整数倍。

13.4.5 默认临时表空间

默认临时表空间是指一旦该数据库启动则默认使用该表空间作为默认的临时表空间，用于存放用户会话数据如排序操作。默认临时表空间可以在创建数据库时创建，此时使用指令 DEFAULT TEMPORARY TABLESPACE，也可以在数据库创建成功后创建，此时需要事先建立一个临时表空间，再使用 ALTER DATABASE DEFAULT TEMPORARY TABLESPACE 指令更改临时表空间。

如果在数据库创建时，没有建立临时表空间，数据库创建成功后也没有创建临时表空间且更改默认设置，则 Oracle 把 SYSTEM 表空间设为默认临时表空间，并且认为这是不合理的行为，并把这个信息记录在告警文件 ALERT.LOG 文件中，告警文件信息如图 13-3 所示。由于临时表空间的区段使用很频繁，会不断地出现磁盘碎片，这样对 SYSTEM 表空间的使用效率有很大影响，所以必须创建临时表空间。在 10g 和 11g 版本中都会有这样的告警。

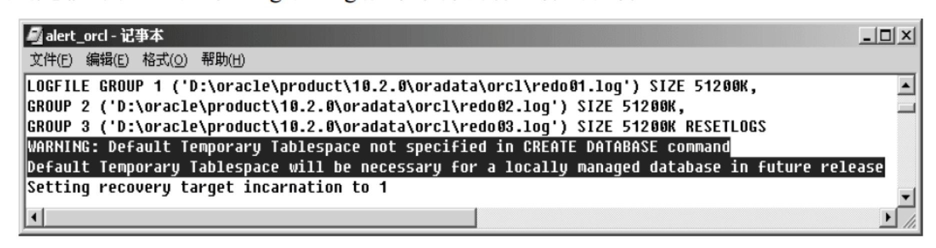


图 13-3 告警文件记录的未设置临时表空间信息

下面，我们先查看下当前数据库的默认临时表空间是那个表空间，使用静态数据字典 DATABASE_PROPERTIES，它有 3 个列属性，分别是属性名（PROPERTY_NAME），属性值（PROPERTY_VALUE）和描述信息（DESCRIPTION），如例子 13-20 所示。

例子 13-20　查看当前数据库的默认临时表空间

```
SQL> col property_name for a30
SQL> col property_value for a20
SQL> col description for a40
SQL> select *
  2  from database_properties
  3* where property_name like 'DEFAULT%'

PROPERTY NAME                   PROPERTY V          DESCRIPTION
---------------------------------------------------------------------------
DEFAULT_TEMP_TABLESPACE TEMP    Name of default     temporary tablespace
```

输出显示当前的默认临时表空间为 TEMP，而在 Oracle 11g 中输出略有不同，如例子 13-21 所示。

例子 13-21　在 Oracle 11g 中查看当前数据库的默认临时表空间

```
SQL> select *
  2  from database_properties
  3* where property name like 'DEFAULT%'

PROPERTY NAME            PROPERTY VALUE   DESCRIPTION
---------------------------------------------------------------------------
```

```
DEFAULT TEMP TABLESPACE    TEMP         Name of default temporary tablespace
DEFAULT_PERMANENT_TABLESPACE USERS     Name of default permanent tablespace
DEFAULT_TBS_TYPE               SMALLFILE    Default tablespace type
```

在例子 13-21 中，输出的最后两行说明已增加 UERS 为默认永久表空间。用户创建的表或索引如果没有指定表空间则默认存储在 USERS 表空间中，而且默认的表空间类型为 SMALLFILE（小文件类型，和 Oracle11g 中的大对象文件类型相对应）。

在 13.5.4 节中，我们已经创建了一个临时表空间 USER_TEMP，下面我们演示如何将临时表空间切换到 USER_TEMP，在生产数据库中，可能会出现当前的临时表空间不能满足应用需求的情况，DBA 可以创建相应的临时表空间，而后切换为当前使用的临时表空间。切换方法如例子 13-22 所示。

例子 13-22　切换临时表空间

```
SQL> alter database default temporary tablespace user temp;

数据库已更改。
```

输出显示已成功更改默认临时表空间，我们下例来验证更改结果。

```
SQL> select *
  2  from database_properties
  3  where property_name like 'DEFAULT%';

PROPERTY NAME             PROPERTY V  DESCRIPTION
--------------------------------------------------------------------------
DEFAULT_TEMP_TABLESPACE  USER_TEMP   Name of default temporary tablespace
```

此时，当前数据库的默认临时表空间为 USER_TEMP。在用户需要时，默认临时表空间可以随时使用例子 13-22 的指令 ALTER DATABASE DEFAULT TEMPORARY TABLESPACE 更改，一旦更改，则所有的用户将自动使用更改后的临时表空间为默认临时表空间。

下面是管理默认临时表空间的一些约束。

- 不能删除一个当前使用的默认临时表空间。

在切换到一个新的临时表空间前，当前的默认临时表空间无法删除，如当前数据库的默认临时表空间为 USER_TEMP，如果想删除该表空间，则提示错误如例子 13-23 所示。

例子 13-23　删除当前使用的临时表空间

```
SQL> drop tablespace user temp;
drop tablespace user temp
*
ERROR 位于第 1 行:
ORA-13906: 不能删除默认的临时表空间
```

想删除当前使用的默认临时表空间，必须建立一个新的临时表空间，并且切换到该新建立的临时表空间。

- 不能把默认临时表空间的空间类型改为 PERMANENT。即不能把默认临时表空间改为一个永久 PERMANENT 表空间。

- 不能把默认临时表空间置为脱机状态。

表空间处于脱机状态的目的是使得使用这些表空间的应用无法再使用它们，从而完成一些诸如脱机备份、维护的任务。但是类似的操作不会涉及临时表空间，所以 Oracle 设计成不能把默认临时表空间置为脱机状态，否则会提示如下错误，如例子 13-24 所示。

例子 13-24　将默认临时表空间脱机

```
SQL> alter tablespace user_temp offline;
alter tablespace user temp offline
*
ERROR 位于第 1 行:
ORA-03217: 变更 TEMPORARY TABLESPACE 无效的选项
```

13.4.6　创建大文件表空间

大文件表空间是在 Oracle 10g 中提出来的，大文件表空间由一个大文件组成，而不是由多个传统的小文件组成，这使得 Oralce 有能力创建和管理大文件。正是大文件表空间和大文件的一一对应特性使得表空间成为磁盘空间管理、备份和恢复的操作对象。

（1）使用大文件表空间的一些限制。

在 Oracle 11g 中只有本地管理的且段空间自动管理的表空间才能使用大文件表空间（Big File Tablespace），我们简称大文件表空间为 BFT。但是对于本地管理的回滚表空间和临时表空间，不要求段空间管理的类型，可以使用 BFT。同时在 Oralce 11g CONCEPT 文档中建议，大文件表空间应该和自动存储管理和逻辑卷管理工具结合使用，这些工具能够支持动态扩展逻辑卷，也支持条带化或支持 RIAD。

使用大文件表空间在数据库开启时对于 DBWR 进程的性能会有显著提高，但是该表空间的大文件会增加该表空间或整个数据库的备份和恢复时间。

（2）使用大文件表空间的优势。

- 只需要创建一个数据文件，大大减少了数据文件的数量，简化了数据文件的管理，显然数据文件的减少使得控制文件的容量也减少，因为控制文件不需要记录大量的数据文件的存储位置信息。
- 大文件表空间的容量比普通表空间要大的多，所以其存储能力显著提高。一个普通的表空间最多可以用 1024 个数据文件，而一个大文件表空间只包含一个文件，但是此文件的容量上限是普通数据文件的 1024 倍，所以大文件表空间和普通表空间的容量是一样的。但是由于每个数据库最多使用 64k 个表空间，所以使用大文件表空间使得数据库总容量比使用普通表空间是要大得多。根据块的大小，大文件表空间的容量可以为 138，如果最大的数据块为 32KB，则数据库最大容量可以达到 8EB。

（3）创建大文件表空间。

创建大文件表空间有 3 种方法，下面我们依次介绍这 3 种方法。

- 在创建数据库时，定义大文件表空间并把它作为默认表空间。

我们给出创建数据库时设置大文件表空间的语句，如下所示。

```
SQL> create database
  1  set default bigfile tablespace tbs_name
  2  datafile 'd:\bigfile_tbs\bfile_tbs01.dbf' size 2G
```

一旦创建了默认表空间为大文件表空间类型，则以后创建的表空间都为大文件表空间，否则需要手工修改这个默认设置。

- *在数据库成功创建后，使用 CREATE TABLESPACE BIGFILE 创建大文件表空间，如例子 13-25 所示。*

例子 13-25　创建大文件表空间 BIGFILETBS

```
SQL> create bigfile tablespace bigfiletbs
  2  datafile 'd:\bigfile tbs\bfile tbs01.dbf' size 2G;

表空间已创建。
```

在成功创建了大文件表空间后，我们再通过例子 13-26 验证该表空间的一些属性信息。

例子 13-26　查询表空间的数据文件属性信息

```
SQL> select tablespace name,file name,bytes/(1024*1024*1024) G
  2* from dba_data_files

TABLESPACE NAME   FILE NAME                                              G
--------------------------------------------------------------------------
USERS
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF      .004882813
SYSAUX
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF    .244140625
UNDOTBS1
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF    .034179688
SYSTEM
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF     .46875
EXAMPLE
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF    .09765625
BIGFILETBS          D:\BIGFILE_TBS\BFILE_TBS01.DBF                      2

已选择 6 行。
```

我们看到表空间 BIGFILETBS 的大小为 2GB，唯一的数据文件位于目录 D:\BIGFILE_TBS 下，文件名为 BFILE_TBS01.DBF。

我们再看一下表空间 BIGFILETBS 的区段管理方式和段空间管理方式，如例子 13-27 所示。

例子 13-27　查询表空间 BIGFILETBS 的区段管理方式和段空间管理方式

```
SQL> select tablespace name,initial extent,contents,extent management,segment
space management
  2  from dba_tablespaces
  3  where tablespace_name like 'BIG%';
```

```
TABLESPACE_NAME      INITIAL_EXTENT CONTENTS  EXTENT_MAN SEGMEN
-------------------- -------------- --------- ---------- ------
BIGFILETBS                    65536 PERMANENT  LOCAL      AUTO
```

大文件表空间 BIGFILETBS 的初始区段（INITIAL_EXTENT）大小为 64KB，而区段管理方式（EXTENT_MAN）为本地管理方式（LOCAL），段空间管理方式为自动管理方式（AUTO）。

- 通过改变默认表空间为大文件表空间使得后来创建的表空间都为大文件表空间。

Oracle 允许动态地改变当前数据库的默认表空间的类型为大文件表空间或普通表空间。我们用例子说明如何把数据库的默认表空间类型改为大文件表空间类型，如例子 13-28 所示。

例子 13-28　把数据库的默认表空间类型改为大文件表空间类型

```
SQL> alter tablespace set default bigfile tablespace
```

（4）更改大文件表空间的大小。

在大文件表空间创建后，可以根据需要修改表空间的大小，有两种方式实现大文件表空间的容量的修改。

- 在 ALTER TABLESPACE 指令中使用 RESIZE 子句，如例子 13-29 所示。

例子 13-29　更改大文件表空间的尺寸

```
SQL> alter tablespace bigfiletbs resize 4G;

表空间已更改。
```

我们通过例子 13-30 验证修改结果。

例子 13-30　查询更改后的大文件表空间 BIGFILETBS 的大小

```
SQL> col file_name for a30
SQL> select tablespace_name,file_name,bytes/(1024*1024*1024) G,autoextensible
  2  from dba_data_files
  3* where tablespace_name like 'BIG%'

TABLESPACE_NAME          FILE_NAME                          G    AUT
------------------------------                        ------------------------------
---------------------------------------- -----------
BIGFILETBS               D:\BIGFILE_TBS\BFILE_TBS01.DBF     4    NO
```

从例子 13-30 的输出可以看出，大文件表空间 BIGFILETBS 的大小已经修改为 4GB，注意该表空间的自动扩展方式为 NO，意味着该表空间不能自动扩展。下面介绍第二种修改大文件表空间的方法。

- 在 ALTER TABLESPACE 指令中使用 AUTOEXTEND ON 子句，如例子 13-31 所示。

例子 13-31　修改大文件表空间大小为自动扩展

```
SQL> alter tablespace bigfiletbs autoextend on next 1G;

表空间已更改。
```

再通过例子 13-32 验证修改结果。

例子 13-32　验证修改文件自动扩展的结果

```
    SQL>        select        tablespace_name,file_name,bytes/(1024*1024*1024)
G,autoextensible
      2  from dba_data_files
      3  where tablespace_name like 'BIG%';

    TABLESPACE_NAME           FILE_NAME                          G   AUT
    ------------------------------------------------------------ -------------
    BIGFILETBS                  D:\BIGFILE_TBS\BFILE_TBS01.DBF   4   YES
```

我们看到该表空间的大小为 4GB，当空间不足时的扩展方式为自动扩展，因为 AUTOEXTENSIBLE 的值为 YES。

如果不知道当前数据库的默认表空间的类型可以使用如例子 13-33 所示的方法查看。

例子 13-33　查询当前数据库的默认表空间的类型

```
    SQL> col property_name for a20
    SQL> col property_value for a20
    SQL> col description for a30
    SQL> select *
      2  from database_properties
      3* where property_name ='DEFAULT_TBS_TYPE'

    PROPERTY_NAME       PROPERTY_VALUE       DESCRIPTION
    ------------------------------------------ ----------------------------
    DEFAULT_TBS_TYPE     SMALLFILE            Default tablespace type
```

我们从输出可以判断当前数据库的默认表空间类型为 SMALLFILE（即小文件表空间），也就是普通表空间。

13.5　表空间管理

本节讲表空间的脱机管理和只读管理，脱机与只读是表空间的两种状态，在脱机状态下，用户或应用程序无法访问这些表空间，此时可以完成一些如脱机备份等操作，处于只读状态的表空间，用户或应用程序可以访问这些表空间，但是无法更改表空间中的数据，如果一个表中的数据不会变化，属于静态数据，这样就可以把相应表空间改为只读，只读表空间不产生变化的数据。

下面我们依次讲解脱机管理和只读管理。

13.5.1　脱机管理

脱机管理的表空间无法实现数据访问，在 Oracle 数据库中不是所有的表空间都可以设为脱机管理，其中包括 SYSTEM 表空间，有活跃还原段的表空间和默认临时表空间。

表空间一般是处于联机状态的，此时用户可以访问表空间中的数据，但是有些情况下需要将表空间置于脱机状态，这些情况包括：

- 允许用户访问数据库的一部分，而某些空间不允许用户访问。
- 执行脱机的表空间备份。
- 在数据库打开时，恢复表空间或表空间中的数据文件。
- 在数据库打开时，移动表空间中的数据文件。

当一个表空间处于脱机状态时，Oracle 不允许执行任何的 SQL 语句，用户试图访问存储在该表空间中的对象会报错。当表空间脱机或联机时，这个事件会记录在数据字典和控制文件中。如果关闭数据库时，数据库处于脱机状态，则当数据打开时依然保持脱机状态。



Oracle 实例有时会自动切换表空间到脱机状态，如当数据库写进程尝试向一个表空间中的数据文件写数据，而尝试失败时，则自动将该表空间切换到脱机状态。

下面通过例子说明如何将一个表空间置为脱机状态，在笔者的数据库上曾经创建了 LIN 表空间，我们先查看该表空间的信息，如例子 13-34 所示。

例子 13-34　查看表空间 lin 的状态

```
SQL> select status,contents,logging
  2  from dba_tablespaces
  3  where tablespace name = 'LIN';

STATUS    CONTENTS  LOGGING
--------- --------- ---------
ONLINE    PERMANENT LOGGING
```

从输出看出该表空间处于联机状态，是永久表空间，用于存储用户表或索引等数据库对象。为了后面演示的需要，我们再看看该表空间中是否表存在，如例子 13-35 所示。

例子 13-35　查看表空间 LIN 是否存在

```
SQL> select table name,owner
  2  from dba tables
  3  where tablespace_name = 'LIN';

TABLE NAME     OWNER
------------------------------------------
EMPLOYEES       SCOTT
```

可见，表空间 LIN 存储了一个表，表名为 EMPLOYEES，所属用户为 SCOTT。下面演示如何将表空间 LIN 设置为脱机状态，如例子 13-36 所示。

例子 13-36　将表空间 LIN 设置为脱机状态

```
SQL> alter tablespace lin offline;

表空间已更改。
```

下面通过例子 13-37 查看修改结果。

例子 13-37　查询表空间 LIN 是否脱机

```
SQL> select status,contents,logging
  2  from dba_tablespaces
  3  where tablespace_name = 'LIN';

STATUS    CONTENTS  LOGGING
--------- --------- ---------
OFFLINE   PERMANENT LOGGING
```

此时，看到该表空间处于离线状态，我们需要确定该表空间中的数据文件的状态，我们用例子 13-38 演示。

例子 13-38　查询表空间 LIN 中的数据文件的状态

```
SQL> select file#,name,status
  2  from v$datafile
  3  where file#>10
  4  ;

FILE# NAME                                STATUS
---------- ------------------------------ ----------------------------
11 D:\TEMP\LIN.DBF                        OFFLINE
```

例子 13-37 和例子 13-38 说明，表空间 LIN 和其中的数据文件都处于脱机状态，这样用户就无法查询该表空间中的数据库对象，否则会提示错误，我们查询表空间 LIN 中的表 EMPLOYEES，看错误提示，如例子 13-39 所示。

例子 13-39　查询表空间 LIN 中的表 EMPLOYEES

```
SQL> show user
USER 为"SYS"
SQL> select *
  2  from scott.employees;
from scott.employees
          *
ERROR 位于第 2 行:
ORA-00376: 此时无法读取文件 11
ORA-01110: 数据文件 11: 'D:\TEMP\LIN.DBF'
```

在上例中，我们先查看了当前用户为 SYS，所以在查询表 EMPOYEES 时必须指明该表所属的用户 SCOTT。查询结果说明，无法读取文件 11，此时的 11 为文件号，如在例子 13-38 中所示。并且 Oracle 知道无法读取的数据文件的位置，遇到这种情况，用户可以根据文件信息，查找该文件的状态，很容易确定问题在哪。

13.5.2　只读管理

只读管理就是把表空间设置为只读状态，这样表空间中的数据只能被用户读取，而不能做任何修改或插入操作，在数据库设计时，如果有的数据是静态数据，则可以将存储这些数据的表放在一个表空间中，只读管理的表空间不被重做日志保护，减少重做日志文件的大小。

把一个表空间置为只读状态的指令为 ALTER DATABASE TABLESPACE_NAME READ ONLY。下面通过例子 13-40 演示如何将 USERS 表空间设置为只读状态。

例子 13-40　将 USERS 表空间设置为只读状态

```
SQL> alter tablespace users read only;

表空间已更改。
```

再通过例子 13-41 查看该表空间的状态。

例子 13-41　查看表空间的状态

```
SQL> select tablespace_name,status
  2  from dba_tablespaces;

TABLESPACE_NAME      STATUS
------------------------------------ --------------
SYSTEM               ONLINE
UNDOTBS1             ONLINE
SYSAUX               ONLINE
TEMP                 ONLINE
USERS                READ ONLY
EXAMPLE              ONLINE

已选择 6 行。
```

我们看到该表空间 USERS 的状态 STATUS 为 READ ONLY，已经设置为只读状态。为了验证只读表空间的只读性，我们通过一系列的例子来验证，首先通过例子 13-42 查看表空间 USERS 中 SCOTT 用户的表信息。

例子 13-42　查看表空间 USERS 中 SCOTT 用户的表信息

```
SQL> select owner,tablespace_name,table_name
  2  from dba_tables
  3* where owner = 'SCOTT'

OWNER                TABLESPACE_NAME      TABLE_NAME
------------------------------------------- ----------------------
SCOTT                USERS                DEPT
SCOTT                USERS                EMP
SCOTT                USERS                BONUS
SCOTT                USERS                SALGRADE
```

我看到在 USERS 表空间中有 SCOTT 用户的 4 个表，分别是 DEPT、EMP、BONUS 和 SALGRADE，下面我们查询表 DEPT 的信息，如例子 13-43 所示。

例子 13-43　查询表 DEPT 的信息

```
SQL> select *
  2  from scott.dept;

    DEPTNO  DNAME            LOC
```

```
------------------------------------------------- ----------------
        10  ACCOUNTING      NEW YORK
        20  RESEARCH         DALLAS
        30  SALES            CHICAGO
        40  OPERATIONS       BOSTON
```

显然可以查询该表空间 USERS 中的表数据，下面再修改表空间 USERS 中表 DEPT 的数据，删除 DEPTNO 为 40 的记录。看是否成功，如例子 13-44 所示。

例子 13-44　删除表 DEPT 中 DEPTNO 为 40 的记录

```
SQL> delete from scott.dept
  2  where deptno = 40;
delete from scott.dept
               *
第 1 行出现错误:
ORA-00372: 此时无法修改文件 4
ORA-01110: 数据文件 4: 'D:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF'
```

我们看到，无法删除表空间 USERS 中表 DEPT 中的记录，因为该表空间设置为只读状态，该表空间中的数据是无法做任何更改的。

在需要的时候，可以将一个只读状态的表空间设置为正常状态，即可读可写状态，如例子 13-45 所示。

例子 13-45　将只读状态的表空间 USERS 设置为正常状态

```
SQL> alter tablespace users read write;

表空间已更改。
```

我们通过例子 13-46 验证是否将表空间 USERS 设置为正常状态。

例子 13-46　验证是否将表空间 USERS 设置为正常状态

```
SQL> select tablespace_name,status
  2  from dba tablespaces;

TABLESPACE_NAME      STATUS
-------------------------------- ---------------
SYSTEM                ONLINE
UNDOTBS1              ONLINE
SYSAUX                ONLINE
TEMP                  ONLINE
USERS                 ONLINE
EXAMPLE               ONLINE

已选择 6 行。
```

输出说明表空间 USERS 的状态为 ONLINE，说明已经将表空间 USERS 设置为可读可写的正常状态。

13.6 表空间和数据文件管理

表空间的管理涉及修改表空间的大小、删除表空间以及修改表空间的存储参数等，数据文件逻辑存放在表空间中，管理数据文件涉及修改数据文件的大小、删除表空间中的数据文件、迁移数据文件以及改变表空间的存储路径。下面依次讲解表空间管理和数据文件管理。

13.6.1 修改表空间大小

修改表空间的大小有 4 种方法：

- 第一种是在创建表空间时，使用 AUTOEXTEND ON 子句使得表空间在需要时可以自动扩展；
- 第二种是在创建表空间后使用 ALTER DATABASE DATAFILE file_name AUTOEXTEND ON 修改不能自动扩展的表空间的数据文件；
- 第三种方法是在表空间中增加数据文件；
- 第四种方法是修改数据文件的大小，即重新设置表空间中某个数据文件的大小。

下面通过例子依次演示上述 4 种修改表空间的方法。

（1）首先演示在创建表空间时，使用 AUTOEXTEND ON 子句创建可以自动扩展的表空间，如例子 13-47 所示。

例子 13-47 创建数据文件自动扩展的表空间

```
SQL> create tablespace manager_tbs1
  2  datafile 'd:/tbs manager1/tbs1.dbf'
  3  size 100M
  4  autoextend on;

表空间已创建。
```

在创建表空间 MANAGER_TBS1 时，我们在 DATAFILE 子句后使用了 SIZE 指定该数据文件的大小。使用 AUTOEXTEND 子句指定该数据文件为自动扩展，如例子 13-48 所示。

例子 13-48 查看表空间 MANAGER_TBS1 的数据文件的扩展方式

```
SQL> select file name,tablespace name,blocks,status,autoextensible
  2  from dba_data_files
  3* where tablespace_name like 'MAN%'

FILE NAME                    TABLESPACE NAME  BLOCKS  STATUS      AUT
------------------------------------ ----------- -------------- -------
D:\TBS_MANAGER1\TBS1.DBF  MANAGER_TBS1    13800   AVAILABLE  YES
```

（2）在创建表空间后，使用 ALTER DATABASE 命令修改表空间中的数据文件为自动扩展，如例子 13-49 所示，先创建一个表空间 MANAGE_TBS 再查看该表空间是否可自动扩展。

例子 13-49 创建一个表空间 MANAGE_TBS

```
SQL> create tablespace manage tbs
  2  datafile 'd:\tbs_manager\tbs01.dbf'
  3  size 50M
  4  uniform size 1M;

表空间已创建。

SQL> select tablespace name,file name,autoextensible
  2  from dba data files;

TABLESPACE_NAME  FILE_NAME                                        AUT

---------------------------------------------------------------------------
USERS            D:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF      YES
SYSAUX           D:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF     YES
UNDOTBS1         D:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF    YES
SYSTEM           D:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF     YES
EXAMPLE          D:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF    YES
MANAGE TBS       D:\TBS MANAGER\TBS01.DBF                               NO
已选择 6 行。
```

在例子 13-49 中，我们先创建了一个表空间 MANAGE_TBS 然后查看了该表空间的 AUTOEXTENSIBLE 属性值为 NO，也就是该表空间在空间不足时不能自动扩展，显然这样的表空间缺少灵活性，也对 DBA 维护数据库增加了负担，现在我们使用 ALTER DATABASE 指令修改该表空间中的数据文件为自动扩展，且需要空间时自动扩展 1MB 空间，如例子 13-50 所示。

例子 13-50　使用 ALTER DATABASE 命令修改表空间中的数据文件为自动扩展

```
SQL> alter database datafile
  2  'd:\tbs manager\tbs01.dbf' autoextend on
  3  next 1MB;

数据库已更改。
```

为了确认修改结果，我们使用例子 13-51 验证表空间 MANAGE_TBS 的数据文件是否处于自动扩展模式。

例子 13-51　查询表空间 MANAGE_TBS 的数据文件的自动扩展模式

```
SQL> col file_name for a30
SQL> select tablespace_name,file_name,autoextensible
  2  from dba data files
  3* where tablespace name like 'MAN%'

TABLESPACE_NAME     FILE_NAME                       AUT
------------------------------------------------------ -----------
MANAGE_TBS          D:\TBS_MANAGER\TBS01.DBF        YES
```

我们看到，表空间 MANAGE_TBS 中有一个数据文件，该数据文件的自动扩展属性值为 YES，所以该文件在表空间不足时，可以自动扩展，每次自动扩展的空间大小为 1MB。

（3）在（2）中使用 ALTER DATABASE 命令修改了表空间中的数据文件可以自动扩展，现在我们用另一种方法即在表空间中增加一个数据文件的方式增加表空间容量。如例子 13-52 所示，

向表空间 MANAGE_TBS 中添加一个数据文件。

注意在例子 13-52 中，我们假设表空间 MANAGE_TBS 中的数据文件没有设置为自动扩展。

例子 13-52 向表空间 MANAGE_TBS 中增加了一个数据文件

```
SQL> alter tablespace manage tbs
  2  add datafile 'D:\TBS MANAGER\TBS02.DBF'
  3  size 50MB;

表空间已更改。
```

此时，我们成功向表空间 MANAGE_TBS 中增加了一个数据文件，该文件大小为 50MB，这样通过增加文件的方式增加了表空间的容量。我们通过例子 13-53 验证表空间 MANAGE_TBS 中的数据文件信息。

例子 13-53 验证表空间 MANAGE_TBS 中的数据文件信息

```
SQL> select tablespace name,file name,status,autoextensible
  2  from dba data files
  3  where tablespace name = 'MANAGE TBS';

TABLESPACE_NAME     FILE_NAME                           STATUS      AUT
--------------------
-----------------------------------------------------------------
------------------------
MANAGE_TBS         D:\TBS_MANAGER\TBS01.DBF             AVAILABLE   NO
MANAGE_TBS         D:\TBS_MANAGER\TBS02.DBF             AVAILABLE   NO
```

通过例子 13-53，我们看到表空间 MANAGE_TBS 中有两个文件表空间，MANAGE_TBS 的大小是两个数据文件大小之和。

不论使用哪种方法修改表空间的大小，都不能超过数据文件所在的磁盘空间，所以要定期查看告警日志文件，查看是否有表空间不足的警告，及时处理，否则会造成数据挂起或者数据库关闭。

（4）我们将表空间 MANAGE_TBS 中的数据文件 D:\TBS_MANAGER/TBS01.DBF 修改为 100MB（修改前为 50MB），如例子 13-54 所示。

例子 13-54 修改表空间 MANAGE_TBS 中的数据文件

```
SQL>conn system/oracle
已连接。
SQL> alter database
  2  datafile 'D:\TBS MANAGER\TBS01.DBF' resize 100MB;

数据库已更改。
```

我们使用 ALTER DATABASE 指令修改了表空间 MANAGE_TBS 中的一个数据文件，在修改前我们必须查询该表空间中的数据文件，而后再使用 ALTER DATABASE 指令修改，因为在修改时不会有任何表空间的提示。

在修改了数据文件 D:\TBS_MANAGER/TBS01.DBF 为 100MB 后，我们再用例子 13-55 验证修改结果。

例子 13-55　验证在例子 13-54 中修改的数据文件大小

```
SQL> select tablespace_name,file_name,bytes,status
  2  from dba_data_files
  3* where tablespace_name = 'MANAGE_TBS'

TABLESPACE_NAME   FILE_NAME                      BYTES      STATUS
------------------------------ --------------- ----------------------
MANAGE_TBS        D:\TBS_MANAGER\TBS01.DBF  104857600  AVAILABLE
```

输出中 BYTES 为 104 857 600B（100MB），说明已经成功修改了表空间中的数据文件的大小，即增大了表空间的容量，通过间接的方式修改了表空间的容量。

13.6.2　修改表空间的存储参数

修改表空间的存储参数只对数据字典管理的表空间有效，在 Oracle 11g 中创建的表空间默认都为本地管理的表空间，为了演示如何修改表空间的存储参数，我们使用例子 13-47 修改在 13.5.1 节中创建的数据字典管理的表空间 tianjin_data 的存储参数 MINIMUM EXTENT，即修改该表空间分配的最小 EXTENT 尺寸为 2MB，如例子 13-56 所示。

例子 13-56　修改该表空间分配的最小 EXTENT 尺寸

```
SQL> alter tablespace tianjin_data
  2  minimum extent 2M;
表空间已更改。
```

例子 13-57　修改表空间 tianjin_data 的默认存储子句

```
SQL> alter tablespace tianjin_data
  2  default storage(initial 2M next 2M maxextents 50);
表空间已更改。
```

为了验证修改结果，我们使用例子 13-58 查看修改后的表空间 tianjin_data 的存储参数。

例子 13-58　查看修改后的表空间 tianjin_data 的存储参数

```
SQL> set line 100
SQL>select tablespace_name, initial_extent,next_extent,max_extents,min_extlen
  2  from dba_tablespaces
  3* where tablespace_name = 'TIANJIN_DATA'
TABLESPACE_NAME   INITIAL_EXTENT NEXT_EXTEN MAX_EXTENTS MIN_EXTLEN
----------------------------------- ---------------------- ------------
TIANJIN_DATA          2097132        2097132        50          2097132
```

通过例子 13-58 输出结果，验证了我们成功修改了表空间 tianjin_data 的存储参数。在修改参数时，我们使用 MB 作为单位，而输出中默认使用字节。



在 Oracle 11g 中已经不允许创建数据字典管理的表空间。只有在 Oracle 9i 以及之前的版本可以创建数据字典管理的表空间。

13.6.3 删除表空间

当不需要一个表空间时，可以删除该表空间以释放磁盘空间，删除表空间的语法格式比较简单，如下所示。

```
DROP TABLESPACE tablespace name
[INCLUDING CONTENTS [AND DATAFILES] [CASCADE CONSTRAINTS]]
```

各参数的含义是：

- Tablespace_name：要删除的表空间名。
- INCLUDING CONTENTS：删除表空间中的所有区段（EXTENTS）。
- AND DATAFILES：删除表空间中的数据文件，该文件是一个 Oracle 格式的操作系统文件。
- CASCADE CONSTRAINTS：删除和该表空间中的表相关的引用完整性约束，外部表会引用该表空间中表的唯一键等作为外部表的引用。

下面我们给出一个例子，删除表空间 MANAGE_TBS 并且删除该表空间中的一个数据文件，如例子 13-59 所示。

例子 13-59 删除表空间并且删除该表空间中的数据文件

```
SQL> drop tablespace manage_tbs including contents and datafiles;

表空间已删除。
```

此时，和该表空间相关联的所有数据文件都一同删除掉。读者可以自己验证这个结果。



如果使用 drop tablespace manage_tbs 指令删除该表空间则只是删除了控制文件中指向该数据文件的文件指针，而实际的数据文件不会被删除。

在删除表空间后，该数据库中不再有该表空间的任何数据，数据库不再管理这些数据文件，只读状态的表空间和表空间的区段仍然可以顺利删除，在删除表空间时，Oracle 推荐将表空间置于脱机状态，以避免仍然有事务在操作表空间中的区段。

13.6.4 迁移数据文件

迁移数据文件是指把当前表空间中的数据文件迁移到其他磁盘空间，可以想象在生产数据库中，当一个表空间所在的磁盘满时，为了使数据库系统正常运行，必须将其中的数据文件迁移到其他空闲磁盘。

迁移数据文件主要分两种，即迁移系统表空间中的文件和迁移非系统表空间中的文件，下面我们分别来介绍。

- 迁移系统表空间中的数据文件。

（1）我们先查看当前系统表空间中的数据文件信息，如下所示。

```
SQL> select tablespace name,file name,bytes,status
  2  from dba_data_files
  3  where tablespace_name = 'SYSTEM';

TABLESPACE NAME      FILE NAME                        BYTES       STATUS
-------------------------------------------------------------------------
SYSTEM              F:\ORACLE\PRODUCT\10.2.0\ORADA  503316480  AVAILABLE
                    TA\ORCL\SYSTEM01.DBF
```

这里的目的是确定系统表空间 SYSTEM 中的数据文件位置，FILE_NAME 属性值给出了文件的具体路径和文件名，F:\oracle\product\10.2.0\oradata\orcl\system01.dbf。

（2）使用具有 DBA 权限的用户登录数据库服务器，然后关闭数据库。

```
SQL> conn system/oracle as sysdba
已连接。
SQL> shutdown immediate;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
```

（3）复制 SYSTEM 表空间的数据文件到新的目录下，该新目录为 D:\SYSTEM。

此时即可使用操作系统指令复制，也可以使用 SQLPLUS 的 HOST COPY，在 UNIX 系统中为 HOST CP。

```
SQL> host copy F:\oracle\product\10.2.0\oradata\orcl\system01.dbf d:\system
已复制         1 个文件。
```

因为系统表空间 SYSTEM 中的数据文件大小不同，所以需要等待的时间也有差异，如上述输出所示，成功复制了 SYSTEM 表空间中的文件。

（4）打开数据库到 MOUNT 状态，然后使用 ALTER DATABASE RENAME FILE 指令迁移数据文件，这里的作用是告诉数据库做了数据迁移和数据迁移地点。数据库再次打开数据文件时，会知道到哪里打开数据文件。

```
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1350380 bytes
Variable Size             176163764 bytes
Database Buffers          419430400 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
SQL> alter database
  2  rename file 'F:\oracle\product\10.2.0\oradata\orcl\system01.dbf'
  3  to 'd:\system\system01.dbf';

数据库已更改。
```

当打开数据库到 MOUNT 状态时，启动实例但是 Oracle 不会打开数据文件，所以迁移数据文件的指令可以成功执行。

（5）打开数据库，使用指令 ALTER DATABASE OPEN。

```
SQL> alter database open;

数据库已更改。
```

在 Oralce 11g 中，需要先执行介质回复，否则会提示错误要求介质回复。介质回复的指令为 RECOVER DATABASE，然后再使用 ALTER DATABASE OPEN 指令打开数据库。

在迁移了 SYSTEM 表空间的数据文件后，我们通过例子 13-60 验证修改结果。

例子 13-60　验证迁移后的系统表空间

```
SQL> col file_name for a50
SQL> run
  1  select tablespace name,file name,status
  2  from dba_data_files
  3* where tablespace_name like 'SYS%'

TABLESPACE NAME     FILE NAME                                        STATUS
---------------------------------------------------------------------------
SYSAUX      F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF     AVAILABLE
SYSTEM      D:\SYSTEM\SYSTEM01.DBF                                 AVAILABLE
```

此时，系统表空间的数据文件已经迁移到新的磁盘目录下。

- 迁移非系统表空间。

在 Oralce 中，要求这种非系统表空间没有活跃的还原段、临时段、排序段等，这种非系统表空间才可以迁移。下面我们迁移数据库中的 MANAGE_TBS，我们知道该表空间中有一个数据文件 D:\TBS_MANAGER\TBS01.DBF，下面将它迁移到 F:\TBS_MANAGER 磁盘目录下。

（1）把表空间 MANAGE_TBS 设置为脱机。

```
SQL> alter tablespace manage tbs offline;

表空间已更改。
```

（2）复制数据文件到新的磁盘目录下，文件名不变。

```
SQL> host copy d:\tbs manager\tbs01.dbf f:\tbs manager
已复制         1 个文件。
```

（3）使用指令 ALTER DATABASE RENAME DATAFILE 迁移数据文件。

```
SQL> alter tablespace manage tbs
  2  rename datafile 'd:\tbs manager\tbs01.dbf'
  3* to 'f:\tbs_manager\tbs01.dbf'

表空间已更改。
```

（4）把表空间联机。

```
SQL> alter tablespace manage_tbs online;

表空间已更改。
```

经过上述步骤，我们已经迁移了非系统表空间 MANAGE_TBS 中的数据文件到新的磁盘目录下，下面我们再通过例子 13-61 验证修改结果。

例子 13-61　验证对非系统表空间的迁移

```
SQL> select tablespace name,file name,status
  2  from dba data files
  3* where tablespace_name like 'MAN%'

TABLESPACE NAME     FILE NAME                      STATUS
------------------------------------------------------- ----------------
MANAGE TBS          F:\TBS MANAGER\TBS01.DBF       AVAILABLE
MANAGER_TBS1        D:\TBS_MANAGER1\TBS1.DBF       AVAILABLE
```

表空间 MANAGE_TBS 中的数据文件名为 F:\TBS_MANAGER\TBS01.DBF，显然我们成功迁移了数据字典，把表空间 MANAGE_TBS 中的数据文件迁移到了 F 盘的目录下。

13.6.5　数据字典和本地管理的表空间

在 Oracle 8i 中的表空间都是数据字典管理的，而在 Oracle 9i 中表空间即可以是数据字典管理的也可以是本地管理的，可以在二者之间切换。但是在 Oracle 11g 中，创建的所有表空间都默认是本地管理，而且如果 SYSTEM 表空间是本地管理的则无法创建数据字典管理的表空间，从这些变化中，读者或许可以体会到 Oracle 对于性能管理的要求更加严格，而且在版本升级中直接支持性能更高的表空间管理方式，逐渐淘汰数据字典管理表空间的方式。

在 Oracle 9i 中，可以调用 PL/SQL 软件包 DBMS_SPACE_ADMIN 中的过程来改变表的管理方式。将数据字典管理的表空间切换为本地管理的表空间的方式如下：

```
SQL> conn  /as sysdba
已连接。
SQL>EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL(tbs_name)
```

而从本地管理的表空间切换到数据字典管理的表空间的方式如下：

```
SQL>EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL(tbs_name)
```

在 Oracle 11g 中，每个数据库都会创建一个 SYSTEM 表空间，它是在创建数据库时自动创建的，但是管理员可以创建本地管理的 SYSTEM 表空间，如果数据库中的 SYSTEM 表空间为本地管理的，则该数据库中不能创建数据字典管理的表空间，否则会提示如例子 13-52 所示的错误。虽然用户可以通过移动表空间的方式向该数据库中添加数据字典管理的表空间，但是该表空间必须是只读表空间。否则，创建字典管理的表空间时会提示如下错误。

```
SQL>  create tablespace tbs1
  2   datafile 'd:\tbs manager\tbs01.dbf' size 100M
  3  extent management dictionary;
 create tablespace tbs1
*
第 1 行出现错误:
ORA-13913: 无法创建字典管理的表空间
```

13.7　本章小结

本章是内容较为丰富的一章，表空间和数据文件是 Oracle 数据库中非常重要的两个概念，表空间是一个逻辑概念，它和段、区段和数据库块组成了数据库的逻辑结构，而数据文件和操作系统块组成了数据库的物理结构，采用逻辑结构和物理结构的结构模式是 Oracle 为了满足其在不同操作系统之间方便地移植而设计的。

本章讲解了表空间的逻辑结构和物理结构之间的关系，从而理解 Oracle 如何操作数据文件以及操作系统如何管理和操作数据文件。Oracle 把表空间的管理方式分为数据字典管理的表空间和本地管理的表空间，而本地管理的表空间是 Oracle 推荐的方式，在 Oracle 10g 和 Oracle 11g 中是默认的表空间管理方式。表空间的维护是 DBA 的一项重要任务，本章讲解了创建各种不同类型的表空间，如还原表空间、临时表空间和默认临时表空间，创建大文件表空间等，如何将表空间置为脱机状态或只读状态，在删除表空间或表空间备份时，需要表空间为脱机状态，而把不变的数据放在一个只读表空间中会减少还原数据量，从而减少数据库服务器的压力。

表空间创建后，可以通过指令修改表空间的参数，但是只对数据字典管理的表空间有效，用户也可以把表空间中的数据文件迁移到其他磁盘，以减少当前数据文件所在磁盘的压力，在不需要表空间时，可以采用不同的方式删除表空间，删除表空间是需要谨慎对待的，因为一旦删除数据很难恢复。

第 14 章

UNDO表空间管理

还原数据是为了实现数据更改的同时，其他用户或进程可以并发访问正在更新而没有提交的数据。除此之外，还原数据在事务恢复和事务回滚时也发挥作用。本章我们首先介绍 Oracle 引入还原数据的作用，然后详细介绍存储还原数据的还原段的类型，而自动还原段管理是 Oracle 推荐的管理方式，还原段逻辑保存在还原表空间中，所以还原表空间的创建与维护也是本章介绍的重点。

14.1　引入还原段的作用

Oracle 还原段的引入确实解决了修改数据时并行读数据的问题，当用户修改数据时，该数据首先复制到还原段中，一个事务将它需要修改的全部数据放在同一个还原段中，我们总结一下还原段的用途，即事务恢复、事务回滚和读一致性，如图 14-1 所示。

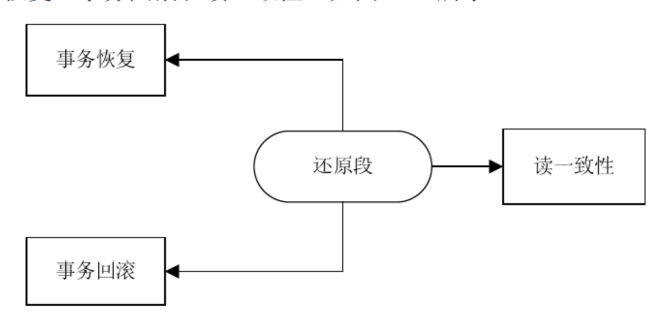


图 14-1　还原段的作用示意图

下面我们依次介绍还原段的几个作用。

- 事务恢复：要实现还原段进行数据恢复，需要将还原段上数据的变化记录在重做日志文件中。一旦某个事务执行期间数据库实例崩溃，再次启动数据库时就需要还原没有提交的数据，恢复成这些数据的原始值。
- 事务回滚：如果用户更改了某行数据，而后恢复修改后且没有提交的数据，使用 ROLLBACK 语句回滚修改的数据，此时 Oracle 数据库服务器就使用还原段中的数据完成数据的回滚操作。
- 读一致性：在用户修改数据期间，如果修改的数据还没有提交，则其他读取该数据的用户

不应该看到这些被修改了且没有提交的数据，而应该看到这些数据的原始值，而这些数据的原始值就放在还原段中。

14.2　还原段如何完成读一致性

先给出一个场景描述：读一致性是相对脏读而言的，表 T 中有 10 000 条记录，获取所有的记录需要 15 分钟的时间，当前时间为 9 点整，用户发出一条 select * from T 命令，该语句在 9 点 15 分完成。当用户执行该语句到 9 点 10 分的时候，另外一个用户发出了一条删除命令，将最后一条记录删除，并且进行了提交。

到 9 点 15 分的时候，如果返回了 9 999 条记录，那么就是脏读；如果是 10 000 条，那么就是读一致性。Oracle 不会出现脏读，提供读一致性，而且没有阻塞 DML 操作。

14.2.1　Oracle 如何实现读一致性

下面我们根据上面的场景继续分析 Oracle 如何实现读一致性的问题。

- 用户在 9 点发出 select 语句的时候，服务器进程会记录 9 点那个时刻的 SCN 号[SCN 号是以时间（timestamp）作为参数的一个函数返回值，调用函数（默认以 timestamp 为参数）随时可以返回这个时刻的 SCN 号，可以使用函数在 SCN 和 timestamp 之间进行转换]，假设该 SCN 号是 SCN9.00，那么 SCN9.00 一定大于等于记录在所有数据块头部的 ITL 槽中的 SCN 号。（如果有多个 ITL 槽，SCN 为最大的那个。）
- 服务器进程扫描 T 表的时候，会把扫描的数据块头部的 ITL 槽中的 SCN 号与 SCN9.00 进行比较，哪个更大。如果数据块头部的 SCN 小于 SCN14.00，那么说明这个数据块在 9 点以后没有更改过，可以直接读取，如果数据块头部的 SCN 号大于 SCN9.00，则说明该数据块在 9 点以后更改过，已经不是 9 点那个时刻的数据了，于是要借助 undo 块。
- 9 点 10 分，用户更改了 T 表的最后一条记录并提交（无论是否提交，只要是更改了 T 表，用户就会去读 undo 数据块），假设被更改的是 N 号数据块，那么 N 号数据块头部的 ITL 槽中记录的 SCN 被修改为 SCN.910，当服务器进程扫描到这个数据块的时候，发现 ITL 槽中的 SCN9.10 大于 SCN14.00，说明该数据块在 9 点以后被更新了，于是服务器进程到 N 号块的头部，找到 SCN9.10 所在 ITL 槽，由于 ITL 槽记录了对应的 undo 块的地址，于是服务器进程找到 undo 数据块，结合 undo 数据块给用户提供读一致性。

14.2.2　读一致性的进一步复杂化分析

下面我进一步复杂化读一致性的问题。这样就可以更深入地分析读一致性对于不同场景的处理。

9 点 10 分，更新了数据并且进行了提交，9 点 11 分又对该数据块进行了更新并且提交（假设数据块只有一个 ITL 槽），那么该 ITL 槽记录的就是 SCN9.11。

这种情况如何处理，关键在于 undo 数据块中，除了记录改变前的数据以外，因为数据块的 ITL 槽也发生了变化，因此也进行了记录，而 ITL 槽中记录了 undo 块的地址 9 点的时候数据

块的 ITL 槽中记录了数据块的 SCN 是 8.50 和对应的 undo 块，9 点 10 分的时候数据库的 ITL 槽中记录了数据块的 SCN 是 9.10 和对应的 undo（undo1）9 点 11 分的时候数据库的 ITL 槽中记录了数据块的 SCN 是 9.11 和对应的 undo（undo2）Undo2 中记录了 9.10 的数据，以及 9.10 数据的 undo 地址 undo1，undo1 中记录了 9 点以前的数据以及 9 点以前的 undo 数据当用户进行查询的时候，服务器进程扫描到 N 号数据块，发现 SCN9.11，大于 SCN9.00，从 ITL 槽中找到 undo 块的地址（undo2），undo2 中记录了改变前的数据和改变前的数据库的 ITL 槽，发现 undo2 对应的 SCN 是 9.10 还是大于 9.00，根据 undo 里面记录的 ITL 槽的改变前信息，继续向前找，找到 undo1，发现 SCN 是 8.50，小于 9.00，使用这个 undo 的数据。

如果在向前寻找的时候，没有找到 SCN 小于 9.00 的数据（因为时间比较长，而且事务已经提交，所以回滚段可能被覆盖），数据库就会出现一个经典的错误 ORA-1555（snapshoot too old），但是不会出现脏读的情况，读一致性示意图如图 14-2 所示。

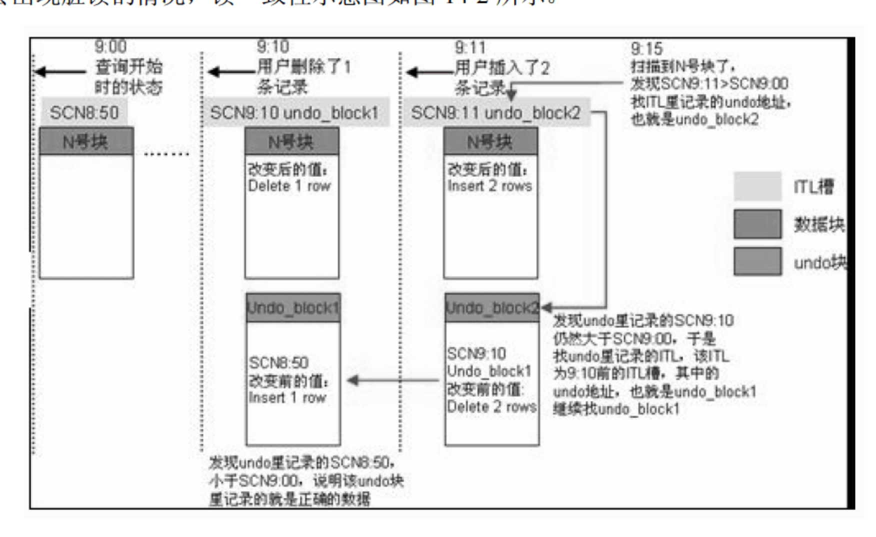


图 14-2　读一致性示意图

最核心的问题是数据块的数据发生改变，ITL 槽也发生改变。这两条信息都会存储到 undo 中。因此只要一个 undo 足够得大，那么一个数据块的所有的 undo 数据块是可以串联起来的。可以从最近一直找到非常远的过去。从数据块开始往前找，一直找到很久以前的 SCN。这个数据块所有的变化都能够找到。ITL 槽中记录着这个数据块的 undo 数据块的地址。

一个数据块中存储很多条数据，update、insert、delete 等 DML 操作，都会影响数据块的 SCN 号。数据块的 SCN 号反映了数据块的变化过程。

14.2.3　读一致性的具体步骤

Oracle 在提供一致性读的过程中，具体的步骤如下：

01 确认读时刻的 SCN。

02 搜寻所有的数据块的 SCN 要求小于读时刻的 SCN。

03 如果搜寻的 SCN 小于读时刻的 SCN，直接读取。

04 如果搜寻的 SCN 大于读时刻的 SCN，根据数据块里面的 ITL 槽里面记录的 undo 信息，找到改变前的数据，如果 SCN 还是大，顺着 ITL 槽信息串联起来的 undo 块继续向前找。

05 如果没有找到小于读时刻的 SCN 的数据块，那么就报错 ORA-155。

事务提交以后，undo 回滚段就可以被覆盖，而且我们在寻找 undo 数据块的时候，这个被寻找的 undo 数据块很可能已经被提交，因此出现 ORA-1555 错误不可避免。Oracle 提供了一个 undo_retention 参数来解决这个问题，在接下来我们会详细介绍该参数的作用。

14.3 还原段的实例恢复与事务回滚

在实例恢复时，Oracle 会读取回滚段的头部记录的事务表，每一个事务是否提交等信息都存储在里面，对于已经提交的事务不做处理，对于未提交的事务完成事务回滚。

根据事务表的信息完成实例恢复。

当需要回滚事务时，由于错误或者 rollback 命令都会产生回滚需求，此时则根据 ITL 槽中记录的 undo 数据块的地址找到 undo 数据块，从而实现恢复数据，即回滚事务。

14.4 UNDO SEGMENT的选择算法

事务 undo segment 发生 DML 操作的时候，服务器进程会选择一个 undo segment，具体算法如下：

- 首先尝试将每一个 undo segment 绑定一个事务，也就是每个 undo segment 上只有一个事务使用。
- 如果不能发现完全空闲的 undo segment，所有的 undo segment 都与事务绑定。
- 系统尝试将脱机的 undo segment 联机。
- 如果没有可用的 undo segment 进行联机，则会尝试创建一个新的 undo segment。
- 如果上面的步骤都没有成功（例如没有可用空间了，不能创建 undo segment），算法会尝试寻找最早使用的 undo segment。这种情况下，不同的多个事务会在同一个相同的 undo segment 里同时进行。
- 每隔 12 个小时会收缩一次，删除那些 idle 状态的 extent。
- DML 操作需要 undo 时，发现空间不够，则会唤醒 SMON 进行一次收缩将 undo segment 里面暂时没有使用的 extent 拿过来使用。

14.5 讨论undo_retention参数

在 Oracle 10g 和 Oracle 11g 中都引入了 UNDO_RETENTION 参数，该参数是一个时间值，说明当还原段中的数据在事务提交后继续保留的时间。该参数默认值为 900 秒，可以根据需要动态更改该参数值。

这个参数以秒为单位，表示当事务提交或者回滚以后，该事务所使用的 undo 块里的数据需要保留多长时间。当保留的时间超过 undo_retention 所指定的时间以后，该 undo 块才能够被其他事务覆盖。当我们使用 AUM 的时候，并且设置了 undo_retention 以后，undo 块的状态就存在以下 4 种。

- Active：表示正在使用该数据块的事务还没有提交或者回滚。
- Inactive：该数据块上没有活动的事务，该状态的 undo 可以被其他事务覆盖。
- Expired：该数据块持续 inactive 的时间超过 undo_retention 所指定的时间。
- Freed：该数据块是空的，从来没有被使用过。

在 AUM 模式中，事务可以在不同的 undo segment 之间动态交换 undo 空间，也就是在不同的 undo segment 里交换 extents。

我们来看一下，当一个事务需要更多的 undo 空间的时候，是如何进行处理的？

获取 undo 表空间里可用的、空的 extents（segment 的最小分配单元是 extent），获取其他 undo segment 里的 expired 状态的 extents。

如果 undo 表空间里的数据文件启用了自动扩展，则数据文件进行自动扩展；如果 undo 表空间里的数据文件没有启用自动扩展，则获取 undo segment 里的 inactive 状态的 extents；如果还是没有获得可用空间，报空间不足的错误。

如图 14-3 所示，undo 表空间中一共存在 5 个 undo segment，每个 undo segment 包括 7 个 extents。

当使用 US5 的事务需要额外的空间的时候，首先使用两个 F 状态的 extent，如果还不够用，继续使用 US4 里面 3 个 F 状态的 extents，如果还不够用，使用 US3 里面 X 状态的 extents，如果还不够用，继续使用 US1 里面 I 状态的 extent，如果好不够用，报空间不足的错误。

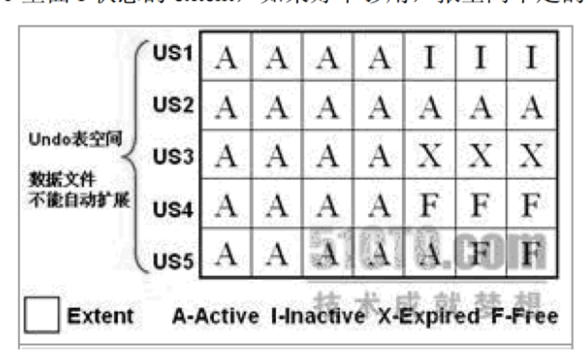


图 14-3　undo 获取可用空间的机制

假设上面的数据文件没有配置自动扩展，则 AUM 里面分配 extent 的方式是高效率的，而且尽量避免使用 I 状态的 extent。

在 Oracle 11g 里面，如果 undo 表空间足够，那么 Oracle 会将 undo 信息保留的时间与当前运行时间最长的查询所需要的时间相同。（一个很好的优势，最大限度的避免了 ORA-1555。）

默认情况下，Oracle 每隔 30 秒就收集统计信息来自动调整 undo retention，收集的信息包括运行时间最长的查询和产生 undo 的速度。我们通过设置 undo_retention 为 0，那么就使用上面的自动调整功能，而且以 900 秒为最低限。如果我们设置了 undo_retention，那么就使用这个参数作为

undo_retention 的值，不再支持动态调整 undo_retention。

如果 undo_retention 设置为 0，则实例会获取运行时间最长的那个查询所需要的时间，例如 N 秒，然后将 undo 信息保留 N 秒，当 undo 表空间尺寸太小时，而不能保留这个最长时间，则尽可能地利用现有空间来让 undo 保留的时间尽可能得长，并不会立刻扩展 undo 数据文件，除非要覆盖的 undo 信息是在 900 秒以内发生的，才会去扩展数据文件。

在还原段中的数据如果保留时间过长，而且修改数据的事务很多，则往往造成还原表空间的不足，所以需要根据业务需要和还原表空间的大小做出判断。

下面演示如何更改参数 UNDO_RETENTION 的大小，如例子 14-1 所示。

例子 14-1　更改参数 UNDO_RETENTION 的值

```
SQL> alter system set undo_retention = 1200;

系统已更改。
```

在更改了参数 UNDO_RETENTION 之后，我们需要验证更改结果，如例子 14-2 所示。

例子 14-2　验证例子 14-1 的更改结果

```
SQL> show parameter undo;

NAME                                 TYPE        VALUE
------------------------------------ ----------- --------------
undo_management                      string      AUTO
undo_retention                       integer     1200
undo_tablespace                      string      UNDOTBS1
```

我们可以看到此时 UNDO_RETENTION 的 VALUE 为 1200，说明我们修改成功。其实，也可以通过数据字典 v$parameter 来查询，该数据字典只有两列：NAME 和 VALUE，下面演示通过例子 14-3 演示如何通过数据字典 v$parameter 查询参数 UNDO_RETENTION 的值。

例子 14-3　通过数据字典 v$parameter 查询参数 UNDO_RETENTION 的值

```
SQL> col name for a20
SQL> col value for a30
SQL> select name,value
  2  from v$parameter
  3  where name = 'undo_retention';

NAME                 VALUE
-------------------- ----------------------
undo_retention       1200
```

该显示结果与例子 14-2 的相同。

14.6　还原段分类

Oracle 数据管理系统将还原段分为两大类，即系统还原段和非系统还原段，其中系统还原段为系统表空间使用，当系统表空间中的对象发生变化时，这些对象的原始值就保存在系统还原段中，

系统还原段在系统表空间中创建，即可以工作在自动模式下也可以工作在手动模式下。

非系统还原段为非系统表空间如用户表空间等所使用。当一个数据库系统具有非系统表空间时，就需要至少一个非系统还原段或一个自动管理的还原表空间，其中自动管理模式由数据库服务器自动维护，但需要至少一个还原表空间，而手动管理模式需要管理员创建非系统还原段，这些手动的非系统还原段又包括两种类型，即：公有还原段和私有还原段。还原段的分类如图 14-4 所示。

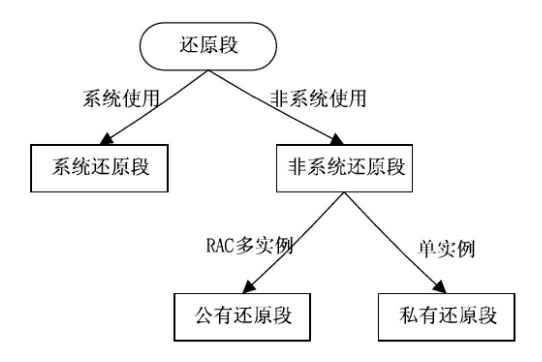


图 14-4　还原段分类示意图

在 Oracle 11g 中实现了还原段的自动管理（实际在 Oracle 9i 以上版本都实现了还原段的自动管理），使用自动管理的方式需要首先创建一个还原表空间，并经还原表空间告诉数据库服务器，之后的维护工作由数据库服务器自动完成。本节我们不设计 RAC 系统，只讲解如何在单实例的数据库系统中维护还原段。

14.7　Oracle的自动还原段管理

在上节我们已经讲了在高版本的 Oracle 数据库中还原段的管理是自动维护的，所以需要为数据库服务器创建一个还原表空间作为存放还原数据的逻辑结构。

在 Oracle 11g 数据库中需要设置两个参数来设置还原段的自动管理，一个为 UNDO_MANAGEMENT，说明还原段的管理方式，它不是一个动态参数，需要在参数文件中修改，然后重新启动数据库方可生效；一个为 UNDO_TABLESPACE，说明还原表空间的名字，该参数是动态参数可以在数据库运行期间动态修改。

在数据库启动后，为了知道当前数据库还原段的管理方式，我们使用例子 14-4 所示的方式来查看。

例子 14-4　查看和还原段相关参数

```
SQL> connect /as sysdba
已连接。
SQL> show parameter undo

NAME                                                    TYPE        VALUE
------------------------------------------------- ------------------
 optimizer undo cost change                             string      11.1.0.6
undo_management                                         string      AUTO
```

```
undo retention                            integer      900
undo_tablespace                           string       UNDOTBS1
```

笔者当前的数据库系统为 11.1.0.6，在例子 14-4 的输出中，我们看到当前的数据库管理方式为自动管理，而还原表空间是 UNDOOTBS1。

当然我们也可以通过参数文件查看还原段的管理信息，如图 14-5 所示。

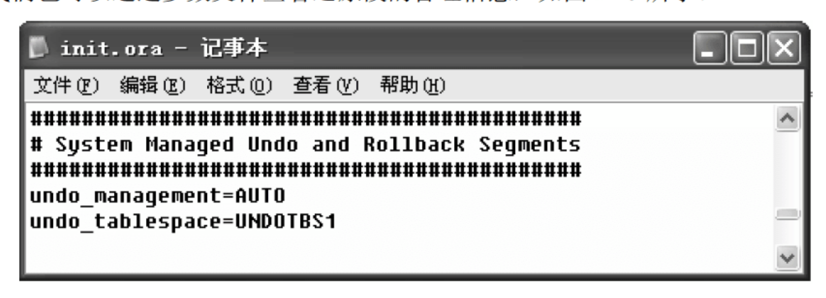


图 14-5　初始化参数文件中的还原段信息

其实，在 Oracle 10g 以及 11g 数据库中，还原段的默认方式都为自动管理方式，而还原表空间为 UNDOTBS1 也是在 Oracle 自动创建的一个还原表空间。当然这个还原表空间可以通过指令修改，如下所示。

```
SQL>ALTER DATABASE SET UNDO_TABLESPACE undo_tablespace_name
```

在以下几节将讲解如何创建还原表空间，如何切换到新的还原表空间以及如何删除还原表空间的问题。

14.8　创建还原表空间

在本节我们讲解如何在创建数据库后创建还原表空间，这也是在实际的生产数据库维护中经常使用的一种创建还原表空间的方式，至于在数据库创建时建立还原表空间可以参考创建 Oracle 数据库一章。

下面我们创建一个还原表空间，使用指令 CREATE UNDO TABLESPACE，如例子 14-5 所示。

例子 14-5　创建一个还原表空间

```
SQL> conn system/oracle@orcl
已连接。
SQL> create undo tablespace my undo
  2  datafile 'd:\undo\my undo.dbf'
  3  size 100MB
  4  autoextend on;

表空间已创建。
```

在例子 14-5 中我们创建了一个还原表空间，其名字为 my_undo，该还原表空间中有一个数据文件，即 d:\undo\my_undo.dbf，大小为 100MB，并且为自动扩展方式，即当还原表空间的空间不足时，该数据文件可以自动扩展。下面我们通过例子 14-6 查看新建的还原表空间的信息。使用数

据字典 DBA_TABLESPACES。

例子 14-6　查看还原表空间的信息

```
SQL> select tablespace name,extent management,contents,logging,status
  2  from dba_tablespaces
  3* where contents ='UNDO'

TABLESPACE NAME         EXTENT MAN CONTENTS  LOGGING   STATUS
---------------------------------------- -------------- --------------
UNDOTBS1                 LOCAL        UNDO     LOGGING   ONLINE
MY_UNDO                  LOCAL        UNDO     LOGGING   ONLINE
```

从上述输出可以看出，还原表空间 MY_UNDO 成功创建，而且处于在线状态，说明随时可以使用切换指令切换将还原表空间切换到 MY_UNDO。EXTENT_MANAGEMENT 为本地管理方式（默认方式），因为 LOGGING 为 LOGGING 所以该还原表空间受到重做日志的保护，可以用于数据库实例恢复。

我们再通过数据字典 DBA_DATA_FILES，查看关于该还原表空间的数据文件的一些信息。读者可以通过 DESC 指令查看数据字典视图 DBA_DATA_FILES 的列属性，来查找自己需要的属性信息。我们给出例子 14-7。

例子 14-7　查看关于还原表空间的数据文件信息

```
SQL> col tablespace_name for a15
SQL> col file_name for a20
SQL> select tablespace name,file name,bytes/1024/1024 MB ,autoextensible
  2  from dba data files
  3* where tablespace name = 'MY UNDO'

TABLESPACE_NAME FILE_NAME                    MB AUT
--------------------------------------------------------- ------ --------
MY_UNDO         D:\UNDO\MY_UNDO.DBF         100 YES
```

例子 14-7 的查询结果说明还原表空间 MY_UNDO 中有一个数据文件，即 D:\UNDO\MY_UNDO.DBF，大小为 100MB，因为 AUTOEXTENSIABLE 为 YES，所以该数据文件为自动扩展方式。



创建还原表空间的指令需要在 DBA 权限的用户模式下操作。

14.9　维护还原表空间

在成功创建还原表空间后，可以通过指令动态的修改还原表空间的一些配置信息，如重命名还原表空间、增加数据文件、将数据文件设置为离线或在线状态等。下面我们依次讲解。

（1）重命名还原表空间

重命名还原表空间使用 RENAME 指令，如我们将 MY_UNDO 重命名为 LIN_UNDO，如例子 14-8 所示。

例子 14-8　重命名还原表空间

```
SQL> alter tablespace my_undo
  2  rename to lin_undo;'

表空间已更改。
```

（2）增加数据文件。

使用 ADD DATAFILE 指令向还原表空间中添加数据文件，以增加还原表空间的容量，如例子 14-9 所示。

例子 14-9　向还原表空间中添加数据文件

```
SQL> alter tablespace my_undo
  2  add datafile 'd:\undo\lin_undo2.dbf'
  3  size 100MB;

表空间已更改。
```

我们向还原表空间 MY_UNDO 添加了一个数据文件，且大小为 100MB，我们通过例子 14-10 验证添加结果。使用数据字典 DBA_DATA_FILES。

例子 14-10　验证例子 14-9 中添加数据成员的结果

```
SQL>select tablespace_name,file_name,bytes/1024/1024 MB ,autoextensible
  2  from dba_data_files
  3* where tablespace_name = 'MY_UNDO'

TABLESPACE_NAME FILE_NAME                                    MB       AUT
--------------------------------------------------------- -------- -------
MY_UNDO         D:\UNDO\MY_UNDO.DBF                          100      YES
MY_UNDO         D:\UNDO\LIN_UNDO2.DBF                        100      NO
```

我们可以看到还原表空间 MY_UNDO 有两个数据文件，其中一个数据文件 LIN_UNDO1.DBF 是我们刚刚添加的，其大小为 100MB。注意此时该文件的 AUTOEXTENSIBLE 为 NO，所以该数据文件填满时将不能自动扩展大小。所以，需要修改该数据文件的参数设置。

（3）将数据文件修改为自动扩展模式。

我们给出例子 14-11 说明如何设置数据文件为自动扩展方式。

例子 14-11　设置数据文件为自动扩展方式

```
SQL> alter database
  2  datafile 'd:\undo\lin_undo2.dbf'
  3  autoextend on;

数据库已更改。
```

现在已经将新添加的数据文件 lin_undo2.dbf 设置为自动扩展了，为了保险起见，我们再通过例子 14-12 验证修改结果。

例子 14-12　验证例子 14-11 的修改结果

```
SQL> select tablespace name,file name,bytes/1024/1024 MB ,autoextensible
  2  from dba_data_files
  3  where tablespace_name = 'MY_UNDO';

TABLESPACE NAME FILE NAME                              MB      AUT
------------------------------
------------------------------------------------------ ---------- ---
MY UNDO        D:\UNDO\MY UNDO.DBF                     100     YES
MY_UNDO        D:\UNDO\LIN_UNDO2.DBF                   100     YES
```

此时，数据文件 D:\UNDO\LIN_UNDO2.DBF 的 AUTOEXTENSIBLE 为 YES，说明已经将该数据文件设置为自动扩展模式了。

我们创建了一个还原表空间 MY_UNDO，并且添加了数据文件以增大该表空间的容量，而且两个数据文件都是自动扩展的方式，所以也方便了数据库的维护。下面我们就分析如何切换还原表空间。

14.10　切换还原表空间

鉴于实际需要，比如还原表空间的磁盘受限，或者还原表空间所在的磁盘过于繁忙，为了减少 I/O 或者避免磁盘空间受限的限制，需要切换还原表空间。在 Oracle 数据库中一个实例只允许有一个活跃的还原表空间，且可以通过动态的方式切换还原表空间。

14.10.1　UNDO 表空间切换示例

我们先查看当前还原表空间的信息，如例子 14-13 所示，然后做切换。

例子 14-13　查看当前还原表空间的信息

```
SQL> show parameter undo;

NAME                                 TYPE        VALUE
------------------------------------------------------------------------------
undo_management                      string      AUTO
undo retention                       integer     900
undo_tablespace                      string      UNDOTBS1
```

我们看到当前的还原表空间为 UNDOTBS1。下面使用例子 14-14 演示如何切换还原表空间。我们切换到新建的还原表空间 MY_UNDO。

例子 14-14　切换到新建的还原表空间 MY_UNDO

```
SQL> alter system set undo_tablespace = my_undo;

系统已更改。
```

在成功执行切换还原表空间的指令后，显示“系统已更改”，为了验证是否成功更改了当前数据库的还原表空间，我使用例子 14-15。

例子 14-15　验证是否成功更改了当前数据库的还原表空间

```
SQL> show parameter undo;

NAME                                 TYPE        VALUE
------------------------------------------------------------------------------
undo_management                      string      AUTO
undo_retention                       integer     900
undo_tablespace                      string      MY_UNDO
```

从例子 14-15 的输出可以看到当前的 UNDO_TABLESPACE 为 MY_UNDO。

14.10.2　UNOD 表空间切换涉及状态

上面做了一个 undo 表空间的切换。

（1）旧的表空间上有事务正在执行，则该旧的表空间变成 pending offline。

（2）用户事务正常运行，切换操作结束，不会等待旧的 undo 表空间的事务结束。

（3）切换以后，所有新的事务所生成的 undo 数据不会存放在旧的 undo 表空间，而是会使用新的 undo 表空间。

（4）Pending offline 状态的 undo 表空间不能被删除。

（5）旧的 undo 表空间上的所有的事务都提交以后，旧的 undo 表空间从 pending offline 状态变成 offline 状态，表空间可以删除。

（6）Drop tablespace undotbs1 相当于 drop tablespace undotbs1 including contents。

（7）如果 undo 表空间包含 inactive 状态的 undo 数据块，不影响被删除，但是可能产生 ORA-1555 错误，因此最好等待超过 undo_retention 以后，再删除表空间。

14.10.3　删除 UNDO 表空间示例

为了演示如何删除一个 UNDO 表空间以及在删除旧的 UNDO 表空间时会遇到的问题，我们先创建一个 UNDO 表空间，然后按照步骤切换并分析中间遇到的问题。

创建 UNDO 表空间 undotbs2。

```
SQL> create undo tablespace undotbs1 datafile '/u01/app/oracle/oradata/PROD/
undotbs02.dbf'
size 10m autoextend on maxsize 100m retention guarantee;

Tablespace created.
```

此时，我们成功创建了一个 UNDO 表空间 undotbs1，当前大小为 10MB，文件为自动扩展，最大尺寸为 100MB，使用了 retention guarantee 参数，以保证在给定时间内的还原记录一定存在不会被覆盖掉，但是该表空间还无法使用，因为 Oracle 只允许有一个 UNDO 表空间。

（2）接着我们启动一个事务，先创建一个表 t，再插入数据，注意此时没有提交，对该表一

致读时依然需要 UNDO 段数据。

```
SQL> create table t (id number,name varchar2(20));

Table created.

SQL> insert into t values (1,'shuhuai');

1 row created.
```

我们没有提交该事务，中间也没有诸如 DDL 语句造成事务隐式地提交。下面切换表空间。

（3）切换 UNDO 表空间。

将当前的表空间 undotbs 切换为 undotbs1，使得将来的事务都是用 undotbs1 作为 UNDO 表空间。

```
SQL> alter system set undo_tablespace=undotbs1;

System altered.
```

我们继续查询当前的 UNDO 表空间以确认修改成功。

```
SQL> show parameter undo;

NAME                                 TYPE                   VALUE
------------------------------------------------------------------------
undo_management                      string                 AUTO
undo_retention                       integer                300
undo_tablespace                      string                 UNDOTBS1
```

接下来尝试删除旧的 UNDO 表空间 undotbs。

（4）删除旧的表空间 undotbs。

```
SQL> drop tablespace undotbs;
drop tablespace undotbs
*
ERROR at line 1:
ORA-30013: undo tablespace 'UNDOTBS' is currently in use
```

此时报错，因为有活动事务。我们回到另一个会话提交事务。

```
SQL> insert into t values (1,'shuhuai');

1 row created.

SQL> commit;

Commit complete.
```

接着我们继续删除该表空间依然报错，如下所示。

```
SQL> drop tablespace undotbs;
drop tablespace undotbs
*
```

```
ERROR at line 1:
ORA-30013: undo tablespace 'UNDOTBS' is currently in use
```

问题涉及一个参数 undo_retention，因为在该参数数值范围内，Oracle 依然会保留旧的 UNDO 记录，所以依然不允许删除该表空间，我们查看参数 undo_rention 的参数值。

```
SQL> sho parameter undo

NAME                                 TYPE                   VALUE
------------------------------------------------------------------------------
undo_management                      string                 AUTO
undo_retention                       integer                5400
undo_tablespace                      string                 undotbs
```

参数 undo_retention 的值为 5400 秒（90 分钟），也就是说对于一个具体的事务在 90 分钟内会保留该事务的 UNDO 数据。所以我们需要等待 90 分钟后再删除旧的表空间，为了完成试验，我们修改该参数为 300 秒，这样等待 5 分钟后即可删除旧表空间 undotbs。下面修改该参数的值为 300 秒，5 分钟后成功删除该表空间。

```
SQL> drop tablespace undotbs;

Tablespace dropped.
```

这里要重述 undo_retention 参数。

Undo_retention 参数不能保证 undo 信息保留足够的时间，Oracle 只是尽量地保证 undo 数据块不被覆盖掉，当空间不够的时候，Oracle 还是会将保留时间小于 undo_retention 的 undo 数据覆盖掉。从 Oracle 10g 开始，我们可以在建立 undo 表空间的时候，设置 retention guarantee 属性，从而避免出现上面的情况。

```
SQL>alter tablespace undotbs1 retention guarantee;
Tablespace altered;
```

也就是说当 undo 数据文件不能扩展，并且 undo 信息不够用时，直接报错，而不是覆盖那些 inactive 而又没有 expired 的 undo 块。

14.11 dba_undo_extents数据字典

使用 dba_undo_extents 数据字典视图可以清楚地看到 UNDO 记录在那个表空间中以及使用的段名和区段的 ID。下面我们给出一个示例。

当前的表空间是 undotbs1，我们继续向表 T 中插入一行数据。

```
SQL> insert into t values (2,'name2');

1 row created.
```

查询使用的回滚段的名字。

```
SQL> select a.username,b.name,c.used_ublk from v$session a ,v$rollname
b,v$transactionc
where a.saddr=c.ses_addr and b.usn=c.xidusn; 2
```

```
    USERNAME                             NAME                    USED_UBLK
    ------------------------------ ------------------------------ ----------
    SYS                                 _SYSSMU114$                1
```

当前使用的回滚段的名字为_SYSSMU114$，接下来使用数据字典视图 dba_undo_extents 查看该回滚段所在的表空间以及区段信息。

```
    SQL>  select  segment_name,tablespace_name,extent_id  from  dba_undo_extents
where segment_name='_SYSSMU114$';

    SEGMENT_NAME                    TABLESPACE_NAME           EXTENT_ID
    ------------------------------ ------------------------------ ----------
    _SYSSMU114$                        UNDOTBS1                        0
    _SYSSMU114$                        UNDOTBS1                        1
    _SYSSMU114$                        UNDOTBS1                        2
    _SYSSMU114$                        UNDOTBS1                        3
```

14.12　本章小结

本章讲解还原数据管理，从 Oracle 引入还原段的目的入手，介绍了还原段的 3 个作用：事务恢复、事务回滚和读一致性。Oracle 按照管理需要将还原段分为系统还原段和非系统还原段，并重点讲了如何实现自动的还原段管理，围绕这一主题，我们详细介绍如何创建和维护还原表空间，其中维护还原表空间涉及重命名、为表空间增加数据文件，设置数据文件为自动扩展方式，同时讲了如何切换还原表空间和删除不用的还原表空间，随后介绍了 UNDO_RETENTION 参数以及使用，最后讨论了几个重要的和还原段相关的数据字典视图。通过这些动态视图读者可以更好地了解还原段的使用情况。

第 15 章

事务（Transaction）

事务是数据库领域一个非常重要的概念，早期的数据库都是联机事务处理系统（即 OLTP），所以对事务都有很好的支持，事务是数据库处理的核心，DBMS 使用事务协调用户的并发行为，减少用户访问资源的冲突。

15.1　Oracle事务的由来

在各种数据库教材中都使用银行取款的例子说明事务的作用，笔者也引用大家习以为常的行为来分析事务的概念和作用。如果读者有在 ATM 机转账的体验，应该很好理解事务的概念。

如果用户 A 要给用户 B 从银行转账 10 000 元，此时我们考虑 ATM 机的行为，把 ATM 机的行为作为一个事务。ATM 机的实施步骤如下：

01 从用户 A 的账户减少 1000 元。

02 向用户 B 的账户增加 1000 元。

上述两个步骤必须都成功执行，如果两个步骤任何一个出现问题，我们说 ATM 机没有正确完成这次转账行为。谁也不希望在自己的账户上白白丢失 10 000 元吧。此时 ATM 机的两个执行步骤是不可分割行为，它要么执行成功，要么不执行（回滚所有更改的数据），ATM 机的两个操作在逻辑上就可以看做一个完整的事务。

15.2　什么是事务

本节我们给出事务的一个更加详细的说明：事务是一组逻辑工作单元，它由一条或多条 SQL 语句组成。一个事务可以在操作的数据库对象上执行一个或多个操作，事务可以作为程序的部分功能而执行。事务开始于一条可执行的 SQL 语句，继续执行事务主体，然后结束于以下的一种情况发生。

- 显示提交 COMMIT：当事务遇到 COMMIT 指令时，将结束事务并永久保存所有更改的数据到数据库文件中。
- 显示回滚 ROLLBACK：当事务遇到 ROLLBACK 指令时，也结束事务的执行，但是此时它回滚所有更改的数据到其原始值，即取消所有更改。

- DDL 语句：一旦用户在使用数据定义语言时，如 CREATE、DROP 等，则之前的所有 DML 语言操作都作为事务的一部分而提交，此时称为隐式提交。
- 正常结束程序：如果 Oracle 数据库应用程序正常结束，如使用 SQL*Plus 工具更改了数据，而正常退出该工具程序，则 Oracle 自动提交事务。
- 非正常地结束程序：当程序崩溃或意外中止时，所有数据更改都被回滚，类似于显示回滚操作的结果，这里是隐式回滚的，因为没有用户参与。

15.3　事务的特点

如果读者学过数据库原理之类的教材，应该会熟悉事务的 4 个特性，简写为 ACID 特性，即 4 个特性的英文首字母。下面详细介绍事务的 4 个特性并说明 Oracle 是如何实现的。

- 原子性（Atomicity）：事务要么执行成功，要么什么也不执行。如果事务执行了一部分而系统崩溃或发生异常，则 Oracle 将回滚所有更改的数据，此时 Oracle 使用还原段管理更改数据的原始值用户事务回滚。
- 一致性（Consistency）：事务必须保持数据库数据在一致状态，如在 SCOTT 用户的 DEPT 表中删除一条记录，但是 EMP 表中存在雇员属于要删除的部门，那就拒绝这样的操作执行。
- 隔离性（Isolation）：隔离性使得多个用户隔离执行实现数据库的并发访问。这种隔离性要求一个事务修改的数据在未提交前，其他事务看不到它所做的更改。Oracle 使用并发控制机制实现事务的隔离性。
- 持久性（Durability）：该特性保证提交的事务永久地保存在数据库中，在 Oracle 数据库中提交的数据并不是立即写入数据文件，而是先保存在数据库高速缓存中，为了防止实例崩溃，Oracle 使用日志优先的方法，首先将提交的数据更改写入重做日志文件，即使实例崩溃也可以在实例恢复，保证事务的持久性。

15.4　事务控制

事务控制使得用户可以控制事务的行为，事务控制有显式和隐式之分，显式控制是指用户使用显式控制指令控制事务，隐式控制是指在某些特定条件下（如 DDL 语句发生时、程序正常退出时）对事务的行为控制。

15.4.1　使用 COMMIT 的显式事务控制

当用户修改数据时，如果想显式提交更改结果，此时可以使用 COMMIT 语句提交更改，因为用户直接使用 COMMIT 指令使得事务提交数据，所以称为显式控制。在用户更改数据时，如果没有提交数据则其他用户看不到该事务所做的数据更改，只有用户提交了更改（无论显示还是隐式的），其他用户才可以看到数据变化。如例子 15-1 所示。我们首先查看 SCOTT 用户的表 DEPT 的内容。

例子 15-1　查看表 DEPT 的内容

```
SQL> select *
  2  from dept;

    DEPTNO  DNAME          LOC
------------------ ------------------------- -------------
        10  ACCOUNTING     NEW YORK
        20  RESEARCH       DALLAS
        30  SALES          CHICAGO
        40  OPERATIONS     BOSTON
```

从输出可以看到表 DEPT 有 4 条记录，下面我们执行一个事务，该事务的作用是从表 DEPT 中删除 DEPTNO 为 40 的记录。

例子 15-2　删除表 DEPT 中的一条记录

```
SQL> delete from dept
  2  where deptno = 40;

已删除 1 行。
```

此时，提示已经删除了 DEPTNO=40 的记录，但是此时无论显式和隐式我们都没有提交数据更改，所以其他用户不应该看到更改后的数据，即其他用户仍然会看到 DEPTNO=40 的记录。我们给出例子 15-3。

例子 15-3　用 SYSTEM 用户登录数据库并查看表 DEPT 的表数据

```
SQL> conn system/oracle@orcl
已连接。
SQL> select *
  2  from scott.dept;

    DEPTNO  DNAME          LOC
----------------- ------------------------- -------------
        10  ACCOUNTING     NEW YORK
        20  RESEARCH       DALLAS
        30  SALES          CHICAGO
        40  OPERATIONS     BOSTON
```

正如我们预料的，SYSTEM 用户可以看到 DEPTNO=40 的记录，这也是 Oracle 实现事务隔离性的体现。下面我们在例子 15-2 执行步骤之后显式提交更改，如例子 15-4 所示。

例子 15-4　显式提交例子 15-2 的更改

```
SQL> commit;

提交完成。
```

此时提交完成，注意此时提交后更改的数据可能并没有写入数据文件，但是由于重做日志的保护，不影响事务的持久性。我们在检验其他用户是否可以看到提交后的数据，即用户应该无法看到 DEPTNO=40 的记录，如例子 15-5 所示。

例子 15-5　用 SYSTEM 用户的登录数据库并查看例子 15-2 提交后的结果

```
SQL> conn system/oracle@orcl
已连接。
SQL> select *
  2  from scott.dept;

   DEPTNO DNAME          LOC
----------------- ---------------------- -------------
       10 ACCOUNTING     NEW YORK
       20 RESEARCH       DALLAS
       30 SALES          CHICAGO
```

在用户提交了数据更改后，我们用 SYSTEM 用户登录数据库，此时可以看到由于事务提交了数据更改，所以其他用户看到的是更改后的数据，即 DEPTNO=40 的记录成功删除了。

用户通过显式的 COMMIT 来提交一个事务，完成事务的持久性，那么当显式 COMMIT 提交前后 Oracle 到底做了哪些工作呢？

在用户显示 COMMIT 提交前，Oracle 数据库内部发生如下行为：

- 在非系统还原段中生成要更改的数据的备份。
- 在重做日志缓冲区创建重做日志选项。
- 在数据库高速缓冲区修改数据（删除或更新）

在用户显式 COMMIT 提交后，Oracle 数据库内部发生如下行为：

- 在重做记录的事务表中标记上已提交事务的 SCN，说明该事务已经提交了。
- LGWR 将事务的重做日志信息和已提交事务的 SCN 号写入重做日志文件。此时，认为提交完成了。
- 释放 Oracle 持有的对更改的数据对象的锁，标记事务完成。

15.4.2　使用 ROLLBACK 实现事务控制

ROLLBACK 回滚所有没有提交的数据更改，Oracle 使用还原段实现回滚功能。由于用户直接使用 ROLLBACK 回滚数据，所以称此为显式的事务控制——事务回滚。我们还是通过例子说明。下面的操作均在 SCOTT 用户模式下实现。我们向表 DEPT 中插入一行记录，如例子 15-6 所示。

例子 15-6　向表 DEPT 中插入一行记录

```
SQL> insert into scott.dept
  2  values(40,'OPERATIONS','BOSTON');

已创建 1 行。
```

此时显示已经成功插入一行记录，其实就是我们在例子 15-2 中删除的那行记录。下面的查询直接在例子 15-6 执行后运行。

例子 15-7　查询插入记录后表 DEPT 中的数据

```
SQL> select *
```

```
  2  from scott.dept;

    DEPTNO DNAME                    LOC
------------------ ------------------------- -------------
        40 OPERATIONS               BOSTON
        10 ACCOUNTING               NEW YORK
        20 RESEARCH                 DALLAS
        30 SALES                    CHICAGO
```

我们看到，查询结果说明已经向表中插入了数据，该行记录显示在第一行。如果此时用户要撤销刚才的插入操作，则可以直接输入 ROLLBACK 指令结束刚才插入记录的事务。

例子 15-8　回滚例子 15-6 中的插入数据记录事务

```
SQL> rollback;

回退已完成。
```

提示回退完成，此时 Oracle 使用重做表空间中的重做记录来恢复数据。我们查看下回滚事务的结果。如例子 15-9 所示。

例子 15-9　查询表 DEPT 在事务回滚后的数据

```
SQL> select *
  2  from scott.dept;

    DEPTNO DNAME                    LOC
------------------ ------------------------ -------------
        10 ACCOUNTING               NEW YORK
        20 RESEARCH                 DALLAS
        30 SALES                    CHICAGO
```

从输出可以看出，没发现 DEPTNO =40 的那行记录，说明事务回滚成功。

15.4.3　程序异常退出对事务的影响

在事务执行过程中，程序会发生异常如实例崩溃等，此时事务结束并回滚所有的数据更改。我们使用 SQL*Plus 的工具登录数据库并执行一个事务，然后不是正常退出而是关闭 DOS 对话框模拟程序异常，看事务如何处理。如例子 15-10 所示打开 SQL*Plus 并登录数据库。

例子 15-10　打开 SQL*Plus 并登录数据库到 SCOTT 模式

```
SQL> conn scott/oracle@orcl
已连接。
SQL>
```

此时，我们执行一个事务，在表 DEPT 中增加一条记录，如例子 15-11 所示。

例子 15-11　向表 DEPT 添加一条记录

```
SQL> insert into dept
  2  values(50,'MARKETING','BOSTON');
```

```
已创建 1 行。
```

此时成功向表 DEPT 中添加一条记录，然后不提交更改而是关闭 DOS 窗口，即关闭如图 15-1 所示的窗口。

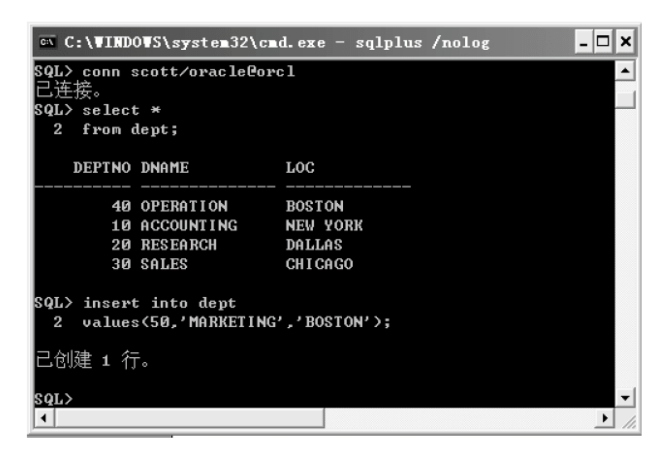
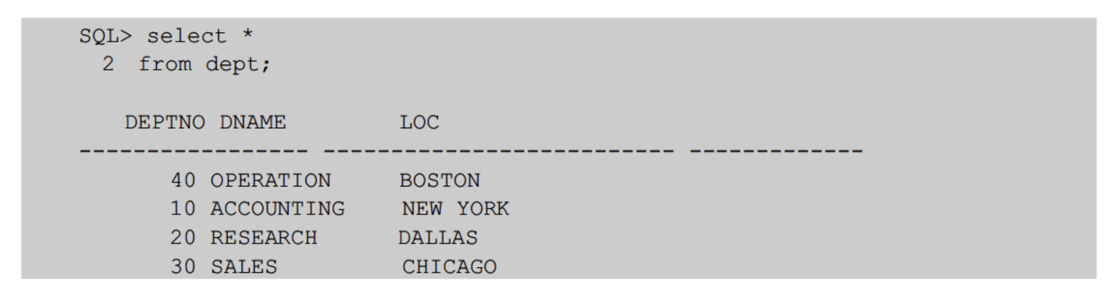


图 15-1　向表 DEPT 中插入一行记录

然后重新登录数据库到 SCOTT 模式，再查询表 DEPT 中的数据，看是否成功添加了一条记录，该记录的 DEPTNO = 50，如例子 15-12 所示。

例子 15-12　查询程序异常退出后插入操作的事务是否成功



```
SQL> select *
  2  from dept;

    DEPTNO  DNAME           LOC
----------------- ------------------------- -------------
        40  OPERATION       BOSTON
        10  ACCOUNTING      NEW YORK
        20  RESEARCH        DALLAS
        30  SALES           CHICAGO
```

我们再重新向表 DEPT 插入一条记录，和图 15-1 中插入的记录一样，不过这次，我们使用指令 EXIT 正常退出 SQL*Plus，然后再登录数据库查看是否成功插入数据，即事务是否隐式提交。

例子 15-13　向表 DEPT 插入一条数据并正常退出程序

```
SQL> insert into dept
  2  values (50,'MARKETING','BOSTON');

已创建 1 行。

SQL> exit
```

然后，再使用 SCOTT 用户登录数据库，查询表 DEPT 的内容看事务是否成功提交。

例子 15-14　程序正常退出后查询事务的是否成功执行

```
SQL> conn scott/oracle@orcl
```

```
已连接。
SQL> select *
  2  from dept;

    DEPTNO  DNAME                LOC
----------------- -------------------------- -------------
        40  OPERATION            BOSTON
        50  MARKETING            BOSTON
        10  ACCOUNTING           NEW YORK
        20  RESEARCH             DALLAS
        30  SALES                CHICAGO
```

此时，表 DEPT 的记录中多了第二行记录，即 DEPTNO=50 的记录，该记录是我们在例子 15-13 中插入的，因为正常退出程序所以 Oracle 隐式提交了数据更改。

15.4.4 使用 AUTOCOMMIT 实现事务的自动提交

Oracle 提供了一种自动提交 DML 操作的方式，这样一旦用户执行了 DML 操作如 UPDATE、DELETE 等，数据就自动提交。

例子 15-15 设置数据库服务器的 AUTOCOMMIT 模式

```
SQL> conn system/oracle@orcl
已连接。
SQL> set autocommit on;
```

在执行 set autocommit on 指令后，不会有任何提示，不过当执行如 DELETE 操作时，数据更改会自动提交。

例子 15-16 在自动提交模式下执行事务

```
SQL> delete from scott.dept
  2  where deptno=50;

已删除 1 行。

提交完成。
```

我们删除 SCOTT 用户的表 DEPT 中 DEPTNO = 50 的记录，此时，我们没有使用显式提交，也没有任何隐式提交的事件发生，此时处于自动提交模式，所以事务自动提交，提示提交完成，这与例子 15-2 不同。

如果不需要自动提交可以关闭自动提交，Oracle 默认是非自动提交。

例子 15-17 关闭自动提交

```
SQL> set autocommit off;
SQL> delete from scott.dept
  2  where deptno = 40;

已删除 1 行。
```

此时，没有提示“提交完成”，说明成功关闭了自动提交。为了方便于以后的数据操作，我

们使用 ROLLBACK 回滚事务。

例子 15-18　回滚事务

```
SQL> rollback;

回退已完成。
```

事务回滚后再查询表 DEPT 的数据，看 DEPTNO= 40 的记录是否存在。

例子 15-19　查询事务回滚后表 DETP 中的数据

```
SQL> select *
  2  from scott.dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        40 OPERATION      BOSTON
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
```

我们看到表 DEPT 中在例子 15-17 中删除的记录（DEPTNO=40）由于事务回滚，依然存在。

15.5　本章小结

本章只讲了事务的概念，事务是一个逻辑工作单元，由一条或多条 SQL 语句组成，事务作用于 Oracle 对象上，如查询表、建立表空间等，由于早期的数据库都是 OLTP 系统所以对事务有很好的支持。事务的 4 个特性（ACID），即原子性、一致性、隔离性和持久性，需要读者认真理解体会，尤其是理解 Oracle 是如何实现其事务的这 4 个特性的。

事务控制使得用户可以更加自由地控制事务的行为，Oracle 模式处于非自动提交模式，即用户事务中执行了 DML 操作后并不是提交该数据，而是需要用户显示或隐式的提交数据更改。在没有提交数据更改时，ROLLBACK 指令可以使得事务回滚。另外在 Oracle 程序非正常退出或实例崩溃时，事务会自动回滚，如果程序正常退出时，事务隐式提交。

第 16 章 角色管理

本章我们讲角色的管理，角色是 Oracle 引入的权限的集合，通过将各种权限赋予角色，使得权限的赋予和回收非常方便，尤其是对于一个大的数据库系统中有很多不同的数据库用户，使用角色可以很好地减少 DBA 的用户权限的管理。

16.1 什么是角色

角色是数据库各种权限的集合，使用角色可以方便地管理数据库特权，角色可以赋予其他用户也可以赋予其他角色，图 16-1 形象地说明了角色的作用。

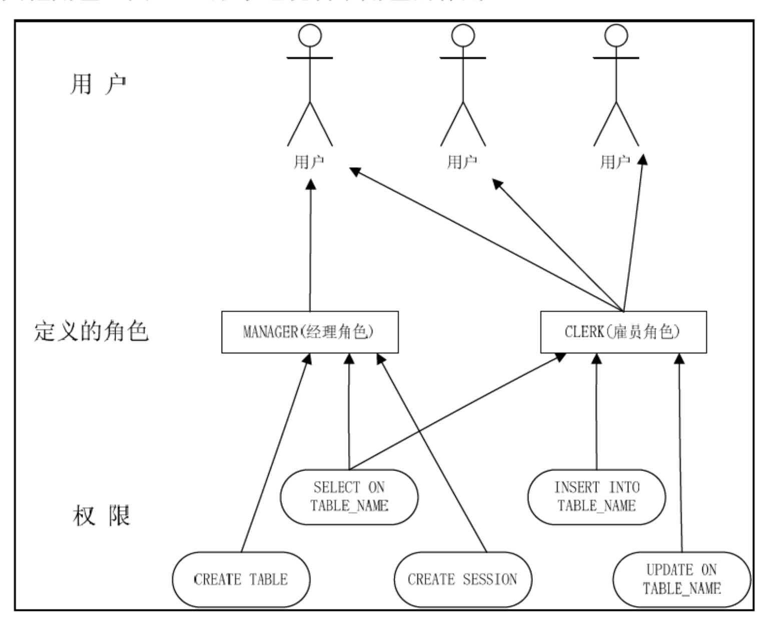


图 16-1　角色、用户和特权的关系图

显然，图 16-1 中有两个角色，而每个角色有不同数目和类型的权限，这些角色又可以赋予不同的用户，这样就方便了权限的管理，作为经理的角色（MANAGER）具有创建会话，创建表以及查询某个表的权限，而雇员（CLERK）角色具有查询表，更新表的权限。然后将 CLERK 角色和 MANAGER 角色赋予第一个用户，此时第一个用户就具有了图 16-1 中的所有权限，而把 CLERK

角色赋予第 2 和第 3 个用户。

使用角色可以减少给用户授予权限的操作次数，同样减少了修改多个用户的权限的操作次数，比如系统上有 10 个用户，需要分别赋予这些用户 10 个权限，如果不使用角色则需要 10×10=100 次操作，而如果使用角色将其作为 10 个特权的集合，再分别赋予 10 个用户，这样就需要 11 次操作，显然极大地减少了数据库操作，提高了 DBA 管理数据库的效率，减少了出错机会。

下面总结一下角色的特点：

- 使用 GRANT 和 REVOKE 授予和回收权限。
- 可以授予任何用户或角色，但是不能赋予角色自己或循环赋予。
- 角色包含系统权限和对象权限。
- 允许启动或关闭赋予用户的角色。
- 允许使用密码启动一个角色。
- 角色名是唯一的，不能和已存在的用户名和角色名相同。
- 角色不被任何人拥有，也不属于任何模式。

角色的描述存储在数据字典 DBA_ROLES 中，如下所示。

```
SQL> select *
  2  from dba roles;

ROLE                          PASSWORD
--------------------------------------------- ------------------
CONNECT                        NO
RESOURCE                       NO
DBA                            NO
SELECT CATALOG ROLE            NO
EXECUTE CATALOG ROLE           NO
...........
```

既然角色有以上的特点，那么我们就从其特点出发总结角色的优点。

- 使得权限的管理更方便，角色赋予多个用户，使得相同的授权很容易实现，而如果需要修改这些用户的权限，只要修改角色就可以修改所有用户的权限。
- 动态的权限管理，一旦角色中的某个权限给修改，则所有的被授予该角色的用户都自动获得修改的权限，并且立即生效。
- 权限可以激活和关闭，使得 DBA 可以方便地选择是否使用赋予用户的角色，临时地关闭或开启角色的使用。
- 可以通过操作系统授权角色，即角色可以通过操作系统指令或工具指定将角色赋予用户。
- 提高性能，使用角色减少了数据字典中授权记录的数量，通过关闭角色使得在语句执行过程中减少了权限的确认。

16.2 创建角色

在描述了什么是角色，总结了角色的好处之后，我们开始创建角色，首先看创建角色的语法

格式，如下所示。

```
CREATE ROLE role name [NOT IDENTIFIED|IDENTIFIED {
BY password | EXTERNALLY | GLOBALLY | USING package}]
```

下面详细介绍各个参数的含义。

- role_name：角色名字。
- NOT IDENTIFIED::在激活角色时不需要密码验证。
- IDENTIFIED：在激活角色时需要密码验证。
- BY password：设置激活角色的验证密码。
- USING package：创建应用角色，该角色只能由应用通过授权的 package 激活。
- EXTERNALLY：说明角色在激活前，必须通过外部服务如操作系统或第三方服务授权。
- GLOBALLY：当使用 SET ROLE 激活角色时，用户必须通过企业路径服务授权来使用角色。

下面我们创建两个角色，注意此时的用户必须具有创建角色 CREATE ROLE 的权限，要创建的 3 个角色分别为 mk_clerk、at_clerk 和 manager，如下所示。

例子 16-1　创建角色 mk_clerk，该角色不需要任何口令标识

```
SQL> connect system/oracle@orcl
已连接。
SQL> create role mk clerk;

角色已创建。
```

例子 16-2　创建角色 at_clerk，该角色在激活时需要口令标识

```
SQL> create role at clerk
  2  identified by rmb;

角色已创建。
```

例子 16-3　创建角色 manager，该角色使用外部服务来标识

```
SQL> create role manager
  2  identified by externanlly;

角色已创建。
```

上面 3 个例子都成功创建了角色，那么如何查看我们在上面创建的角色呢？Oracle 提供了数据字典 dba_roles。如下例所示。

例子 16-4　通过数据字典 dba_roles 来查看角色信息

```
SQL> select *
  2  from dba roles
  3  where role in ('MK CLERK','AT CLERK','MANAGER');

ROLE                           PASSWORD
------------------------------------------------ --------
AT CLERK                        YES
MANAGER                         YES
```

```
    MK_CLERK                            NO
```

显然数据字典 dba_roles 仅记录了角色名和角色密码信息，角色 AT_CLERK 和角色 MANAGER 都需要密码，在 Oracle 9i 中角色 MANAGER 的 PASSWORD 属性为 EXTERNAL，在 Oracle10g 中统一设置为 YES，只说明需要密码验证而不再细分密码类型。角色 MK_CLERK 在创建时没有使用 IDENTIFIED BY 选项，所以激活该角色时不需要密码验证。

16.3　修改角色

角色可以修改，但是 Oracle 只允许修改它的验证方法，修改角色的语法格式如下所示。

```
    ALTER ROLE role {NOT IDENTIFIED | IDENTIFIED {BY password | USING package |
EXTERNALLY | GLOBALLY}}
```

参数的含义如下所示，同创建角色时的参数含义相同，修改角色时只能修改验证方法。

- ROLE
- NOT IDENTIFIED
- BY password
- EXTERNALLY
- GLOBALLY

下面我们修改在 16.2 节创建的 3 个角色。

例子 16-5　修改角色 MK_CLERK 的验证方法为外部标识

```
SQL> alter role mk_clerk
  2  identified by externally;

角色已丢弃。
```

例子 16-6　将角色 AT_CLERK 的验证方法改为不需要任何标识

```
SQL> alter role at clerk
  2  not identified;

角色已丢弃。
```

例子 16-7　将角色 MANAGER 的验证方法改为需要密码标识

```
SQL> alter role manager
  2  identified by rmb;

角色已丢弃。
```

上述 3 个例子已经成功修改了角色的验证方法，下面通过数据字典 DBA_ROLES 来验证修改结果，如例子 16-8 所示。

例子 16-8　通过数据字典 DBA_ROLES 来验证角色的修改结果

```
SQL> select *
```

```
  2  from dba roles
  3  where role in ('MK_CLERK','AT_CLERK','MANAGER');

ROLE                              PASSWORD
-------------------------------------------------------- ---------------------------
AT_CLERK                           NO
MANAGER                           YES
MK_CLERK                          YES
```

输出结果说明角色 AT_CLERK 不需要密码验证，而角色 MK_CLERK 和 MANAGER 需要密码验证，其中角色 MK_CLERK 需要外部服务方式验证，如通过操作系统验证等。

16.4　赋予角色权限

我们知道角色是权限的集合，所以在创建了角色后，就需要将各种权限赋予该角色，赋予角色权限的语法格式如下所示。

```
GRANT  权限 | 角色 TO  角色名
```

在使用该指令向角色授权时，可以将权限或角色赋予其他角色。

下面我们将 CREATE SESSION、SELECT TABLE、CREATE VEIW 权限赋予角色 AT_CLERK，如例子 16-9 所示。

例子 16-9　为角色 CLERK 赋予权限

```
SQL> grant create session,select any table,create view
  2  to at clerk;

授权成功。
```

该例中将权限赋予了角色 AT_CLERK，我们通过数据字典 ROLE_SYS_PRIVS 验证我们的授权结果，如例子 16-10 所示。

例子 16-10　验证角色 AT_CLERK 的权限信息

```
SQL> select *
  2  from role sys privs
  3  where role = 'AT_CLERK';

ROLE                              PRIVILEGE                         ADM
------------------------------------------------------------- --------------
AT CLERK                           CREATE SESSION                     NO
AT_CLERK                           CREATE VIEW                        NO
AT_CLERK                           SELECT ANY TABLE                   NO
```

输出说明角色 AT_CLERK 具有了 3 个权限（PRIVILEGE），并且每个权限的 ADM 选项值都为 NO，说明该角色不能再其将拥有的权限赋予其他用户或角色。

下面我们将 CREATE ANY TABLE 的权限和角色 AT_CLERK 赋予角色 MANAGER，如例子 16-11 所示。

例子 16-11　将权限和角色 AT_CLERK 授予角色 MANAGER

```
SQL> grant create any table,at clerk
  2  to manager;

授权成功。
```

同样我们通过数据字典 ROLE_SYS_PRIVS 检查角色 MANAGER 具有的权限，如例子 16-12 所示。

例子 16-12　查看角色 MANAGER 具有的系统权限

```
SQL> select *
  2  from role_sys_privs
  3  where role = 'MANAGER';

ROLE                          PRIVILEGE                  ADM
------------------------------------------------------------ ------------
MANAGER                        CREATE ANY TABLE           NO
```

或许输出的结果令读者失望，我们命名授予了用户权限 CREATE ANY TABLE 和角色 AT_CLERK，但是没有角色 AT_CLERK 中的系统权限，这就说明使用该数据字典只能查询直接赋予该角色的权限，而无法查看授予该角色的角色信息。那么如何验证将角色 AT_CLERK 赋予角色 MANAGER 呢？Oracle 提供了一个数据字典 DBA_ROLE_PRIVS，如例子 16-13 所示。

例子 16-13　通过数据字典 DBA_ROLE_PRIVS 查看角色授予信息

```
SQL> select *
  2  from dba_role_privs
  3  where granted role = 'AT CLERK';

GRANTEE                    GRANTED ROLE             ADM  DEF
-------------------------------------------------------------- -------
MANAGER                       AT_CLERK               NO   YES
SYSTEM                        AT_CLERK                YES  YES
```

在上述输出中，说明角色 AT_CLERT 授予了用户 SYSTEM（该用户创建了角色）和角色 MANAGER。用户 SYSTEM 可以继续将该角色授予其他用户或角色，因为该行的 ADM 为 YES，并且角色 AT_CLERK 为用户 SYSTEM 的默认用户，因为该行的 DEF 为 YES。对于角色 MANAGER 而言，它不能再将角色 AT_CLERK 赋予其他用户或角色，因为该行的 ADM 为 NO。在下节我们再详细介绍 ADM 这个参数。

16.5　赋予用户角色

我们知道角色是权限的集合，所以在创建了角色后，就需要将各种角色赋予用户或所有用户（PUBLIC），将角色赋予用户的语法格式如下所示。

```
GRANT role [, role ] ……
TO    {user | role | public} | [, { user | role | public }] ……
```

```
[WITH ADMIN OPTION]
```

参数含义如下所示。

- role：赋予用户的角色名（如多个角色，则用逗号隔开）。
- user：被赋予角色的用户（如多个用户用逗号隔开）
- public：将角色赋予所有用户
- WITH ADMIN OPTION：被赋予该角色的用户或角色可以继续将该角色赋予其他用户或角色。

在将角色赋予用户前，我们先创建两个用户 CLERK 和 MYMANAGER，如例子 16-14 所示。

例子 16-14　创建用户 CLERK 和 MYMANAGER

```
SQL> create user clerk
  2  identified by cl12#;

用户已创建。
SQL> create user mymanager
  2  identified by my12#;

用户已创建。
```

我们通过数据字典 DBA_USERS 验证是否成功创建用户，如例子 16-15 所示。

例子 16-15　验证是否成功创建用户

```
SQL> select username,created
  2  from dba_users
  3* where username in ('CLERK','MYMANAGER')

USERNAME                       CREATED
------------------------------------------------ --------------
MYMANAGER                      17-9 月 -13
CLERK                          17-9 月 -13
```

上述输出说明已经创建了两个新用户 MYMANAGER 和 CLERK，下面我们计划将 MANAGER 角色赋予该用户。我们尝试使用新用户 MYMANAGER 登录数据库，如下所示。

例子 16-16　尝试使用新用户 MYMANAGER 登录数据库

```
SQL> connect mymanager/my12#@orcl
ERROR:
ORA-01045: user MYMANAGER lacks CREATE SESSION privilege; logon denied


警告：您不再连接到 ORACLE。
```

提示用户 MYMANAGER 缺少 CREATE SESSION 权限，拒绝登录，显然这样的结果也在预料之中，因为新用户没有任何权利。下面将角色 MANAGER 赋予用户 MYMANAGER 并带有 WITH ADMIN OPTION 选项。

例子 16-17　将角色 MANAGER 赋予用户 MYMANAGER

```
SQL> grant manager
  2* to mymanager with admin option

授权成功。
```

我们使用 DBA_ROLES_PRIVS 来验证授权结果，如例子 16-18 所示。

例子 16-18　验证 MANAGER 角色授予的用户信息

```
SQL> select *
  2  from dba role privs
  3  where granted role ='MANAGER';

GRANTEE                        GRANTED_ROLE            ADM DEF
---------------------------------------------------- ------- ---------
MYMANAGER                      MANAGER                 YES YES
SYSTEM                         MANAGER                 YES YES
```

从输出可以清晰地看出角色 MANAGER 被赋予用户 MYMANAGER，并且该用户具有将角色 MANAGER 继续授权的能力，因为 ADM 值为 YES，角色 MANAGER 为用户 MANAGER 的默认角色，因为 DEF 值为 YES。

下面我们使用 MYMANAGER 用户登录数据库，如例子 16-19 所示。

例子 16-19　使用被赋予 MANAGER 角色的用户 MYMANAGER 登录数据库

```
SQL> connect mymanager/my12#@orcl
已连接。
```

成功连接数据库，说明角色 MANAGER 中的权限起作用，下面我们查询当前用户 MYMANAGER 的会话级权限。

例子 16-20　查看用户 MYMANAGER 的会话级权限

```
SQL> select *
  2  from session privs;

PRIVILEGE
----------------------------------------
CREATE SESSION
CREATE ANY TABLE
SELECT ANY TABLE
CREATE VIEW
```

显然这些权限全部是角色 MANAGER 中的权限，其中 CREATE ANY TABLE 系统权限是单独赋予角色 MANAGER 的，而其他 3 个权限是通过被赋予角色 AT_CLERK 而得到的。

我们知道在例子 16-17 中，赋予用户 MYMANAGER 角色时带有 WITH ADMIN OPTIN 选项，所以可以继续将角色 MANAGER 赋予其他用户。我们演示这个行为。

例子 16-21　在 MYMANAGER 模式下再将角色 MANAGER 赋予用户 CLERK

```
SQL> grant manager
  2  to clerk;
```

```
授权成功。
```

提示授权成功，说明此时用户 CLERK 具有了角色 MANAGER 的所有权限。我们使用 CLERK 用户登录数据库。查询当前用户中角色的授予情况，和当前用户的权限。

例子 16-22　使用新用户 CLERK 登录数据库

```
SQL> connect clerk/cl12#@orcl
已连接。
```

然后查询当前用户中，角色的授予情况。如例子 16-23 所示。

例子 16-23　查看 CLERK 用户的角色信息

```
SQL> select *
  2  from user role privs;

USERNAME                        GRANTED_ROLE           ADM DEF OS_
------------------------------------------------------ -------- ------ ------
CLERK                            MANAGER                 NO  YES NO
```

输出说明用户 CLERK 被授予了角色 MANAGER，该角色不能被该用户继续授权，因为 ADM 的值为 NO，而且该角色不是操作系统验证的，因为参数 OS_为 NO，但是该角色是用户 CLERK 的默认角色，使用用户 CLERK 登录数据库时，该角色自动激活。

其实，读者可以预测此时用户 CLERK 具有同 MYMANAGER 用户一样的会话级权限。读者可以使用数据字典 SESSION_PRIVS 查询，如同例子 16-20，留作读者自己验证。

16.6　默认角色

在 16.4 节，我们看到将角色 MANAGER 赋予用户 MYMANAGER 和用户 CLERK 后，角色 MANAGER 都是用户的默认角色，其实，这是一种默认设置。Oracle 允许使用 ALTER USER 指令修改默认角色。为了演示方便，我们将 AT_CLERK 角色赋予用户 CLERK。

例子 16-24　将角色 AT_CLERK 赋予用户 CLERK

```
SQL> connect system/oracle@orcl
已连接。
SQL> grant at clerk to clerk;

授权成功。
```

我们查询用户 CLERK 被授予的角色信息，使用数据字典 DBA_ROLE_RPIVES。

例子 16-25　查看用户 CLERK 被赋予的角色信息

```
SQL> select *
  2  from dba_role_privs
  3  where grantee = 'CLERK';

GRANTEE                         GRANTED_ROLE           ADM DEF
```

```
-------------------------------------------------------- --------- ---------
CLERK                          MANAGER                    NO   YES
CLERK                          AT_CLERK                    NO   YES
```

我们看到用户 CLERK 被赋予了两个角色，即 MANAGER 和 AT_CLERK，而且都为默认角色。下面我们演示如何将角色 MANAGER 设置为非默认角色，如例子 16-26 所示.

例子 16-26　将角色 MANAGER 设置为非默认角色

```
SQL> alter user clerk default role all except manager;

用户已更改。
```

我们通过例子 16-27 验证修改结果，看是否将角色 MANAGER 设置为非默认角色。

例子 16-27　验证角色 MANAGER 是否为非默认角色

```
SQL> select *
  2  from dba_role_privs
  3  where grantee = 'CLERK';

GRANTEE                         GRANTED ROLE             ADM DEF
------------------------------------------------- --------- ---------
CLERK                          MANAGER                    NO   NO
CLERK                          AT_CLERK                    NO   YES
```

显然，用户 CLERK 的角色 MANAGER 不再是默认角色，因为 DEF 列的值应经为 NO 了。另外，读者可按需要把赋予该用户的角色全部设置为非默认角色，如例子 16-28 所示。

例子 16-28　将用户 CLERK 的所有角色设置为非默认角色

```
SQL> alter user clerk default role none;

用户已更改。
```



此时用户 CLERK 不具有任何角色，也没有任何权利了。我们只有使用 SYSTEM 用户（其他具有相应访问权限的用户也可以）来登录数据库，查看用户 CLERK 的角色授予信息。如例子 16-29 所示。

例子 16-29　使用 SYSTEM 用户的登录数据库并查看用户 CLERK 的角色信息

```
SQL> select *
  2  from dba_role_privs
  3  where grantee = 'CLERK';

GRANTEE                         GRANTED ROLE                     ADM      DEF
--------------------------------------------------------- --------- -------
CLERK                          MANAGER                          NO       NO
CLERK                          AT_CLERK                         NO       NO
```

从输出可以看出用户 CLERK 具有两个角色，对应的 DEF 值都为 NO，说明我们在例子 16-28 中将所有角色设置为非默认角色的指令执行成功。

设置用户 CLERK 的角色中一个或多个为默认角色，如例子 16-30 所示。

例子 16-30　将用户 CLERK 的角色 AT_CLERK 设置为默认角色

```
SQL> alter user clerk default role at clerk ;

用户已更改。
```

此时，将角色 AT_CLERK 设置为用户 CLERK 的默认角色，我们使用数据字典 DBA_ROLE_PRIVS 查看例子 16-30 的修改情况，如例子 16-31 所示。

例子 16-31　验证是否将角色 AT_CLERK 设置为用户 CLERK 的默认角色

```
SQL> select *
  2  from dba_role_privs
  3  where grantee = 'CLERK';

GRANTEE                          GRANTED ROLE                      ADM       DEF
------------------------------------------------------------- --------- -------
CLERK                           MANAGER                            NO        NO
CLERK                           AT_CLERK                           NO        YES
```

显然，输出结果中角色 AT_CLERK 对应的行（第 2 行）属性 DEF 的值为 YES，说明将角色 AT_CLERK 设置为用户 CLERK 的默认角色修改成功。

如果要把授予用户的多个角色都设置为默认角色，可以使用如下的方式实现。我们仍然以 CLERK 用户为例。如例子 16-32 所示。

例子 16-32　将赋予用户的所有角色设置为默认角色

```
SQL> connect system/oracle@orcl
已连接。
SQL> ALTER USER clerk DEFAULT ROLE ALL;

用户已更改。
```

现在输出提示“用户已更改”，说明已经成功将用户 CLERK 的所有角色都设置为默认角色。为了验证这个结果，我们再次使用数据字典 DBA_ROLE_PRIVS 来查看用户 CLERK 的角色信息。

```
SQL> select *
  2  from dba role privs
  3  where grantee ='CLERK';

GRANTEE                          GRANTED_ROLE                 ADM       DEF
-------------------------------------------------------------- -------- ---------
CLERK                           MANAGER                       NO        YES
CLERK                           AT_CLERK                      NO        YES
```

显然，此时角色 MANAGER 也为用户 CLERK 的默认角色了，因为 DEF 值为 YES，使用这种方式设置用户角色为默认角色很方便，尤其对有多个角色的用户而言更是如此。



在将某个角色设置为用户默认角色时，如果该角色是通过其他角色授予该用户的，则该角色不能在 DEFAULT ROLE 子句中使用。

16.7 禁止和激活角色

角色可以禁止和激活，禁止意味着用户不再具有该角色赋予的各种权限，即回收角色具有的权限，而激活意味着赋予用户角色的权限。我们知道用户 CLERK 具有两个角色 MANAGER 和 AT_CLERK。下面我们查看该用户具有的角色权限，如例子 16-33 所示。

例子 16-33 查看用户 CLERK 的用户权限

```
SQL> connect clerk/cl12#@orcl
已连接。
SQL> select *
  2  from session_privs;

PRIVILEGE
----------------------------------------
CREATE SESSION
CREATE ANY TABLE
SELECT ANY TABLE
CREATE VIEW
```

这里再说明一下角色 MANAGER 的权限，该角色通过被赋予 CREATE ANY TABLE 权限和角色 CLERK 而获得权限，而角色 CLERK 通过被赋予 CREATE SESSION、SELECT ANY TABLE 和 CREATE VIEW 来获得权限。这样用户 CLERK 被赋予了两个角色 MANAGER 和 CLERK，这样它就具有了 4 个权限，如例子 16-33 所示。

下面我们演示如何禁止用户的角色，如例子 16-34 所示，禁止用户 CLERK 的所有角色。

例子 16-34 禁止用户的所有角色

```
SQL> set role none;

角色集
```

此时，系统禁止了用户的所有角色，即系统回收了这些角色的权限。下面我们查询当前用户 CLERK 的权限，如例子 16-35 所示。

例子 16-35 在禁止所有角色后查询用户权限

```
SQL> select *
  2  from session_privs;

未选定行
```

可见，用户 CLERK 不具有任何会话权限了，下面我们激活用户 CLERK 的 AT_CLERK 角色。

例子 16-36 激活用户 CLERK 的角色 AT_CLERK

```
SQL> SET ROLE at_clerk ;

角色集
```

现在激活了角色 AT_CLERK，我们查询用户 CLERK 是否具有了角色 AT_CLERK 的权限。

例子 16-37 查询用户 CLERK 是否具有角色 AT_CLERK 的权限

```
SQL> select *
  2  from session_privs;

PRIVILEGE
----------------------------------------
CREATE SESSION
SELECT ANY TABLE
CREATE VIEW
```

该输出和例子 16-35 不同，这里用户 CLERK 具有了角色 AT_CLERK 的权限。下面我们再激活角色 MANAGER。

例子 16-38 激活角色 MANAGER

```
SQL> set role manager;
set role manager
*
第 1 行出现错误:
ORA-01979: 角色 'MANAGER' 的口令缺失或无效
```

没有成功激活，错误提示为缺少口令，读者或许有印象，在创建角色 MANAGER 时，我们使用了验证方式，即使用密码验证，参考例子 16-7。我们再次通过密码验证的方式激活角色 MANAGER。

例子 16-39 激活设置了密码验证的角色 MANAGER

```
SQL> set role manager identified by rmb;

角色集
```

此时，激活角色 MANAGER，我们通过数据字典 SESSION_PRIVS 看用户 CLERK 是否具有了 CREATE ANY TABLE 权限（该权限是角色 MANAGER 拥有的），如例子 16-40 所示。

例子 16-40 查看用户 CLERK 的权限

```
SQL> select *
  2  from session privs;

PRIVILEGE
-------------------------------
CREATE SESSION
CREATE ANY TABLE
SELECT ANY TABLE
CREATE VIEW
```

从输出看出用户 CLERK 具有角色 MANAGER 和角色 CLERK 的权限，说明已成功激活了角色 MANAGER。

16.8　回收和删除角色

既然可以赋予用户角色，也可以回收用户角色，Oracle 允许使用 REVOKE 子句回收赋予某一用户的角色。为了演示如何回收角色，我们在系统上查看创建的角色和这些角色赋予的用户信息等。

例子 16-41　查看创建的角色信息

```
SQL> connect system/oracle@orcl
已连接。
SQL> select *
  2  from dba roles
  3  where role in ('AT CLERK','MANAGER');

ROLE                              PASSWORD
----------------------------------------------- ------------------
AT CLERK                           NO
MANAGER                           YES
```

我们看到，创建的角色 AT_CLERK 和 MANAGER 存在，然后我们继续查询这两个角色赋予的用户信息，即哪些用户被赋予了这些角色，如例子 16-42 所示。

例子 16-42　查看角色 AT_CLERK 和 MANAGER 赋予的用户信息

```
SQL> select *
  2  from dba role privs
  3  where granted role in  ('AT CLERK','MANAGER')
  4  order by granted role;

GRANTEE                       GRANTED_ROLE        ADM         DEF
--------------------------------------------------------- ------- -------
CLERK                         AT CLERK            NO          YES
MANAGER                       AT_CLERK            NO          YES
SYSTEM                        AT_CLERK            YES         YES
MYMANAGER                     MANAGER             YES         YES
SYSTEM                        MANAGER             YES         YES

已选择 6 行。
```

从输出可以看出角色 AT_CLERK 赋予了用户 CLERK、角色 MANAGER 和用户 SYSTEM，而角色 MANAGER 赋予了用户 MYMANAGER 和用户 SYSTEM。注意，角色 AT_CLERK 和 MANAGER 是在用户 SYSTEM 下创建的，所以该用户自动被授予该角色，并且为默认角色。

下面我们回收用户 CLERK 的 AT_CLERK 角色，如例子 16-43 所示。

例子 16-43　回收用户 CLERK 的 AT_CLERK 角色

```
SQL> revoke at clerk from clerk;

撤销成功。
```

然后，我们检验角色 AT_CLERK 是否已经从用户 CLERK 回收，如例子 16-44 所示。

例子 16-44　验证是否正确回收用户 CLERK 的角色 AT_CLERK

```
SQL> select *
  2  from dba_role_privs
  3  where granted_role = 'AT_CLERK'
  4  ;

GRANTEE                        GRANTED_ROLE                   ADM DEF
------------------------------ ------------------------------ --- ---
MANAGER                        AT_CLERK                       NO  YES
SYSTEM                         AT_CLERK                       YES YES
```

我们看到，在上述从查询的 WHERE 子句中，限制查询 GRANTED_ROLE 为 AT_CLERK 的用户或角色信息，在 GRANTEE 列中没有了用户 CLERK，说明角色 AT_CLERK 已经从用户 CLERK 回收。



在回收角色时，可以同时回收多个角色，角色名之间使用逗号隔开，也可以同时从几个用户回收一个或多个相同的角色，用户名之间使用逗号隔开。

但是角色 AT_CLERK 依旧在系统中存在，只是没有授予用户使用，如果不需要可以删除该角色，如例子 16-45 所示。

例子 16-45　删除角色 AT_CLERK

```
SQL> drop role at_clerk;

角色已删除。
```

我们通过数据字典 DBA_ROLES 查看是否成功删除角色 AT_CLERK，如例子 16-46 所示。

例子 16-46　验证角色 AT_CLERK 是否删除

```
SQL> select *
  2  from dba_roles
  3  where role in ('AT_CLERK','MANAGER');

ROLE                          PASSWORD
--------------------------------------------- ------------------
MANAGER                        YES
```

显然，ROLE 列中没有了角色 AT_CLERK，说明成功删除角色 AT_CLERK。

有这样一种情况就是需要将一个角色授予所有的用户，比如查询普通的表权限的角色，如果将这样的角色赋予所有的用户，也是很耗时的，Oracle 使用 PUBLIC 代表所有用户，可以使用如下的方式将一个或多个角色赋予所有用户，如例子 16-47 所示。

例子 16-47　将角色授予所有用户

```
SQL> grant manager to public;

授权成功。
```

此时，我们将 MANAGER 角色赋予了所有用户。然后通过数据字典 DBA_ROLE_PRIVS 查看角色 MANAGER 的赋予用户信息。

例子 16-48　查看角色 MANAGER 的赋予用户信息

```
SQL> select *
  2  from dba role privs
  3* where granted_role in ('MANAGER','PUBLIC')

GRANTEE                            GRANTED ROLE        ADM        DEF
---------------------------------------------- -------- ---------
MYMANAGER                          MANAGER             YES        YES
SYSTEM                             MANAGER             YES        YES
PUBLIC                             MANAGER             NO         YES
```

此时，我们看到角色 MANAGER 赋予了 3 个用户，即 MYMANAGER、SYSTEM 和 PUBLIC。那么如何回收授予 PUBLIC 的角色呢，如例子 16-49 所示。

例子 16-49　回收授予 PUBLIC 的 MANAGER 角色

```
SQL> revoke manager from public;

撤销成功。
```

为了验证角色的回收结果，下面我们查询是否成功回收授予 PUBLIC 的角色 MANAGER。

例子 16-50　查询是否成功回收授予 PUBLIC 的角色 MANAGER

```
SQL> select *
  2  from dba_role_privs
  3* where granted role in ('MANAGER','PUBLIC')

GRANTEE                            GRANTED ROLE        ADM        DEF
-------------------------------------------------------- --------- --------
MYMANAGER                          MANAGER             YES        YES
SYSTEM                             MANAGER             YES        YES
```

显然，已经成功从 PUBLIC 回收角色 MANAGER，因为 GRANTEE 列已经没有 PUBLIC 值，但是单独授予用户的 MANAGER 角色不会通过 REVOKE MANAGER FROM PUBLIC 而被回收。如授予用户 MYMANAGER 的角色 MANAGER 就继续保留下来。

16.9　Oracle预定义的角色

除了我们自定义的角色外，Oracle 预定义了一些很有用的角色，比如 DBA 角色、RESOURCE 角色等。下面我们列出这些角色，并分析这些角色作用。

下面是 Oracle 定义的角色列表，这是 Oracle 11g 中系统预定义的角色。

- AQ_ADMINISTRATOR_ROLE：管理 QUEUE 的管理员角色。
- CONNECT：连接数据库权限。
- DBA：数据库管理员权限。

- EXP_FULL_DATABASE：导出数据库权限。
- IMP_FULL_DATABASE：导入数据库权限。
- JAVADEBUGPRIV：调试 Java 程序权限。
- MGMT_USER：创建会话和创建触发器权限。
- OEM_ADVISOR：执行 OEM 顾问的权限。
- OEM_MONITOR：执行 OEM 监视的权限。
- OLAP_DBA：执行联机事务处理时的 DBA 权限。
- OLAP_USER：执行联机事务处理时的 USER 权限。
- RECOVERY_CATALOG_OWNER：恢复数据字典。
- RESOURCE：创建一系列数据库对象的权限。
- SCHEDULER_ADMIN：管理各种调度的权限，如创建任务、执行程序等。

在 Oracle 11g 中对于预定义的角色安全性有一定程度的加强。在 Oracle 9i 中，CONNECT 角色具有创建视图/表、建立会话、创建同义词/序列号以及创建数据库连接的权利，而在 Oracle 11g 中该角色只具有 CREATE SESSION 权限，显然提高了系统的安全性。并且在 Oracle 11g 中去除了 RESOURCE 角色，而 Oracle 9i 中 RESOURCE 角色和 CONNECT 角色的部分权限统一放入 DBA 角色中。

读者可以通过数据字典 ROLE_SYS_PRIVS 查询系统的预定义角色。我们先使用 SYSTEM 用户登录。

例子 16-51 使用 SYSTEM 用户登录数据库

```
SQL> connect system/oracle@orcl as sysdba
已连接。
```

然后，我们查询 EXP_FULL_DATABASE 角色有哪些权限。

例子 16-52 查询角色 EXP_FULL_DATABASE 的权限信息

```
SQL> select *
  2  from role_sys_privs
  3  where role ='EXP_FULL_DATABASE'
  4* order by privilege

ROLE                          PRIVILEGE                        ADM
------------------------------------------------------------- --------
EXP_FULL_DATABASE            ADMINISTER RESOURCE MANAGER      NO
EXP_FULL_DATABASE            BACKUP ANY TABLE                 NO
EXP_FULL_DATABASE            EXECUTE ANY PROCEDURE            NO
EXP_FULL_DATABASE            EXECUTE ANY TYPE                 NO
EXP_FULL_DATABASE            READ ANY FILE GROUP              NO
EXP_FULL_DATABASE            RESUMABLE                        NO
EXP_FULL_DATABASE            SELECT ANY SEQUENCE              NO
EXP_FULL_DATABASE            SELECT ANY TABLE                 NO

已选择 8 行。
```

从 PRIVILEGE 列的值可以清楚地看出角色 EXP_FULL_DATABASE 的权限，但是该角色一旦

授予用户，该用户不能继续将该角色授予其他用户或角色，因为 ADM 列的值为 NO。

16.10　本章小结

本章我们讲解了角色管理，角色就是数据库权限的集合，这些权限可以是系统权限，也可以是数据库对象权限，使用角色可以方便对用户授权的管理，减少授权操作。用户可以根据业务需求来设计角色所拥有的权限，再将这些权限使用 grant 子句赋予不同的用户。赋予用户的权限在不需要的时候可以使用 REVOKE 子句回收角色，也可以禁止和激活赋予用户的角色，此时使用 SET ROLE NONE 来禁止所有的角色，使用 SET ROLE role_name 来激活角色。如果角色在定义时使用了密码验证，则使用 SET ROLE role_name INDENTIFIED BY password 的方式激活该角色。

为了管理方便 Oracle 预定了一些角色，如 CONNECT、DBA 角色等。读者可以通过数据字典 ROLE_SYS_PRIVS 查询这些预定义角色，以及这些角色的权限。

第 17 章
管理和维护表

管理和维护表在数据库设计、数据库维护中都有很重要的应用，表是数据库中最基本、最常用的存储数据的存储结构。本章将讲解存储在表中的数据类型，如何创建和维护表，重点介绍了几个重要概念和索引组织表，段空间管理方式、行迁移、 水位线等重要概念，对于表的维护十分重要，在本章最后介绍了表的参数维护和列的维护。

17.1　Oracle基本的数据存储机制——表

表是 Oracle 数据库中最基本的数据存储结构，表是一种逻辑结构，数据在表中以行和列的形式存储。在创建表时，用户需要设定表名、表内各列的列名和各列的数据类型及数据宽度。数据类型包括 VARCHAR2，DATE，NUMBER 或 BLOG 等。在表中同一行的所有列信息叫做记录。下面主要介绍 Oralce 的数据类型。

17.1.1　数据的存储类型

在 Oracle 的相关文档中，Oracle 定义了 3 种数据类型，分别是标量数据类型、集合数据类型和关系数据类型，这些数据类型可以用户定义表中列的数据类型。

图 17-1 是 Oracle 中数据类型的关系图。

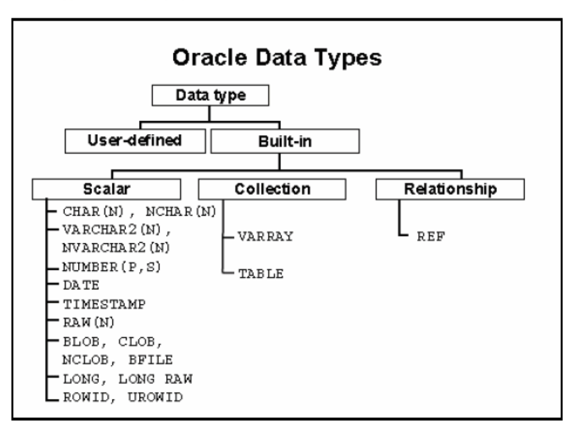


图 17-1　Oracle 的数据类型

下面依次介绍标量数据类型（Scalar）、集合数据类型（Collection）和关系数据类型（Relationships）。

（1）标量数据类型。

- VARCHAR2(size)和 NVARCHAR2(size): 变长字符型数据

首先，该数据类型存储变长的字符数据，在使用该数据类型定义数据时，该数据的存储区大小是不固定的，依据存储数据的长度进行动态分配存储区。参数 size 是该变量存储的最大的字符数，该值最大为 17000。size 的最小或默认值都是 1。一般在定义该数据类型时，都要指定该长度值，即指定 size 值。NVARCHAR2(size)的不同之处在于它支持全球化数据类型，支持定长和变长字符集。

- CHAR(size) NCHAR(size): 定长字符型数据

该数据类型一旦定义，则存储该变量的存储区的大小就固定下来。显然在存储区分配上它没有 VARCHAR2(size)和 NVARCHAR2(size)数据类型具有动态性，但是在实际应用中，如果可以预测到一个变量存储的字符数量，且数量不是很大，则最好还是使用定长字符型数据来定义该变量，这样可以提高存储的效率。因为使用变长字符型数据要不断地计算存储的数据长度，再分配存储数据块，会消耗计算资源。定长字符型数据的最小值和默认值都为 1 个字符，而最大值为 2000。NCHAR(size)的不同之处在于它支持全球化数据类型，支持定长和变长字符集，此时定长字符型数据的最小和默认值都为 1 个字节。

- DATE: 日期型数据

ORACLE 服务器使用 7 个定长的存储区存储日期型数据，它可以月，年，日，世纪，时，分和秒。日期型数据的取值范围从公元前 17712 年 1 月 1 日到公元 9999 年 12 月 31 日。

- NUMBER(P,S): 数字型数据

参数 p 指十进制数的中长度，s 为该十进制数小数点后的位数，如 NUMBER(10,2)表示该数字型数据的中长度为 10 位，而小数后为 2 位。其中参数 p 的最大值为 38，最小值为 1，而参数 s 的最大值为 1217，最小值为-817。

- CLOB 或 LONG: 用于存储大数据对象，该对象为定长的字符型数据，如学术论文或个人简历等。对于 CLOB 数据类型的列的操作不能直接使用 Oracle 数据库指令，需要一个 DBMS_LOB 的 PL/SQL 软件包来维护该数据类型的列。
- BLOB 或 LONG RAW: 存储无结构的大对象，如照片，PPT，二进制图像等。它和 CLOB 数据类型一样，BLOB 数据类型的列的操作只能通过 PL/SQL 软件包 DBMS_LOB 来实现。
- BFILE: 在操作系统文件中存储无结构的二进制对象，显然它是 ORACLE 的外部数据类型，Oracle 无法直接维护这些数据类型，必须由操作系统来维护。
- RAW: 该数据类型使得数据库可以直接存储二进制数据，在计算机之间传输该类型数据时，数据库不对数据做任何转换，所以该数据类型的存储和传输效率较高，RAW 数据类型的最大长度为 2000 个字节。



LONG 和 LONG RAW 数据类型主要用在 Oracle 8 以前的数据库系统。LONG 数据类型完全可以用 Oracle 9i，oracle 10g，oracle 11g 的 CLOG 或 BLOB 数据类型替换。

（2）集合数据类型。

Oralce 集合数据类型包括嵌套表和 VARRY 数据类型。在嵌套表的列值中又包含表，嵌套表中的元素数量没有限制，当然不能超过表所在的表空间的大小，而 VARRY 集合中的元素是有数量限制的。

（3）关系数据类型。

关系类型 REF 指向一个对象，在 Oracle 数据库中一个典型的 REF 类型的对象就是游标 cursor。

17.1.2 行 ID（ROWID）

ROWID 也是一种数据类型，但是这种数据类型是 Oracle 服务器使用并管理的。首先解释 ROWID 的特性，通过特性可以理解 ROWID 的作用。

- ROWID 是数据库中每一行的唯一标识符。
- ROWID 作为列值是隐式存储的。
- ROWID 不直接给出行的物理地址，但是可以用 ROWID 来定位行。
- ROWID 提供了最快速地访问表中行的方法。

虽然 ROWID 是非显式存储的，但是对于每一个表，都可以查询该表中每一行的 ROWID。下面通过例子查看 ROWID，并解释 ROWID 的组成和各组成部分的含义，如例子 17-1 所示。

例子 17-1 登录数据库并查看表 DEPT 的行 ID（ROWID）

```
SQL> conn scott/tiger;
已连接。
SQL> select deptno,dname,loc,rowid
  2  from dept;

DEPTNO DNAME            LOC            ROWID
--------------------------------------------------------------------------
      10 ACCOUNTING     NEW YORK       AAAH17EAAIAAAAByAAA
      20 RESEARCH       DALLAS         AAAH17EAAIAAAAByAAB
      30 SALES          CHICAGO        AAAH17EAAIAAAAByAAC
      170 OPERATIONS    BOSTON         AAAH17EAAIAAAAByAAD
      50 Marcketing                    AAAH17EAAIAAAAByAAE
```

我们选择 DEPTNO 为 20 的行，即输出的第二行的 ROWID 来分析，前 6 位 AAAH17E 为数据对象号，在数据库中每个对象是唯一的。接着 3 位是 AAI 为相对文件号，它和表空间中的一个数据文件对应。接着 17 为 AAAABy 的块号，块号为相对文件中存储该行的块的位置，最后三位 AAB 为行号，行号标识块头中行目录的位置，而使用该行目录的位置可以找到行的起始地址。



本节只要求读者对于 ROWID 有个了解，知道用 ROWID 可以唯一标识一行，Oracle 使用 ROWID 可以快速找到一行数据。

17.2　创建表

本节从实用的角度介绍如何创建一个表以及如何使得创建的表易于管理。我们先介绍 Oracle 创建表的规则，并通过实例说明如何创建一个表。

17.2.1　Oracle 创建表的规则

Oralce 数据库推荐了一些与表相关的标准，读者在实际中最好使用这些标准，对于维护数据库表和顺利建表都很有好处。

- 命名尽量简单，表名要具有一定的意义，即表名要清楚描述表中存储的数据内容，如一个临时员工表的表名为 temp_employees。
- 每个表一个表空间，这样易于管理和维护，对一个表空间的维护不影响其他的表，并且该表空间是本地管理的。
- 使用标准 EXTENT 尺寸减少表空间碎片。
- Oracle 数据库允许表名的最大长度为 30 个字符。

17.2.2　创建普通表

创建数据库的目的是存储数据，而这些数据就存储在表中，表是数据库中最基本的数据存储机制。下面我们使用 CREATE TABLE 指令来创建表。

要创建表用户必须创建表的属性，此时，我们使用 dba 用户登录数据库服务器，如例子 17-2 所示。

例子 17-2　使用 dba 用户登录数据库服务器

```
SQL> conn /as sysdba
已连接。
```

接下来创建一个临时员工表，该员工表属于 SCOTT 用户，并且存储在 USERS 表空间中。如例子 17-3 所示。

例子 17-3　创建一个临时员工表 temp_employees

```
SQL> create TABLE scott.temp_employees
  2      (employee_id          number(17),
  3       employee name        varchar(30),
  17       employee sex         char,
  5       department          varchar(30))
  17       TABLEspace users;

表已创建。
```

一旦使用 DDL 语句创建了表对象，对象的信息如表名，表存储的表空间等将记录在数据字典中，例子 17-3 中创建表的信息将记录在数据字典 dba_tables 中。下面使用例子 17-4 验证是否成功创建该表。

例子 17-4　验证例子 17-3 中是否成功创建表 TEMP_EMPLOYEES

```
SQL> select owner,table name,tablespace name
  2  from dba_tables
  3  where owner = 'SCOTT';

OWNER                    TABLE NAME          TABLESPACE NAME
----------------------------------------------------------------------
SCOTT                     BONUS                  USERS
SCOTT                     DEPT                   USERS
SCOTT                     DEPT TEMP              SYSTEM
SCOTT                     EMP                    USERS
SCOTT                     EMP_TEMP
SCOTT                     ORD                    SYSTEM
SCOTT                     PRODUCT                SYSTEM
SCOTT                     SALGRADE               USERS
SCOTT                     SUPPLIER                SYSTEM
SCOTT                     TEMP_EMPLOYEES         USERS

已选择 10 行。
```

输出结果的最后一行说明，已经成功创建了表 TEMP_EMPLOYEES，该表所属的用户为 SCOTT，而存储该表的表空间为 USERS。



如果在创建表时不指定用户名字，直接写表名，则默认是当前用户创建的表，如果不指定表空间名，则 Oracle 将使用默认表空间创建该表。

在创建表的原则中，Oracle 推荐了一个表最好放在一个表空间而且该表空间是本地管理的（减少维护数据字典的负担），所以如果我们已经创建了一个本地管理的表空间可以使用更多的参数在本地管理的表空间中创建表，在 Oracle 10g 中创建的表空间，本地管理是默认方式。如例子 17-5 所示，我们先创建一个本地管理的表空间 lin，然后再在该表空间中创建一个表。

例子 17-5　创建一个本地管理的表空间 lin

```
SQL> create TABLEspace lin
  2  datafile 'd:\temp\lin.dbf'
  3  size 30M
  17  extent management local
  5  uniform size 1M;

表空间已创建。
SQL> create TABLE scott.employees
  2        (ecode    number(17),
  3         ename    varchar2(25),
  17          eaddress varchar2(30),
  5         ephone   varchar2(15))
  17  storage (initial 100k next 100k pctincrease 0 minextents 1
  7  maxextents 8)
  8* TABLEspace lin
```

```
表已创建。
```

这里需要解释 storage 中的参数含义，initial 指对于该表而言，当表的数据量增加时，需要的自动分配磁盘空间时第一次分配 100k，第二次也是 100k，所分配的最大磁盘为 8 个 EXTENTS。最小为 1 个 EXTENTS，而 PCTINCREASE 是一个权值参数指当第三次为该表增加磁盘空间时，需要按规则计算：$NEXT*(1+PCTINCRASE/100)^{(n-2)}$，其中 n>=3，即如果第三次需要增加磁盘空间时，分配 $100*(1+0/100)^{(3-2)}=100k$，第四次需要增加磁盘空间时，分配 $100*(1+0/100)^{(17-2)}=100k$，可以看出如果选择 PCTINCRASE 为 0，则每次分配的磁盘空间和 NEXT 参数值相同。

下面使用例子 17-6，验证是否成功建立表 employees。

例子 17-6　验证是否成功建立表 employees

```
SQL> select TABLE name,TABLEspace name ,initial extent,next extent
  2  from dba_TABLEs
  3  where owner = 'SCOTT'
  17  and TABLE name = 'EMPLOYEES';

TABLE NAME    TABLESPACE NAME    INITIAL EXTENT    NEXT EXTENT
---------------------------------------------------- -------------------
EMPLOYEES      LIN                1021700           101785717
```

上述输出说明，表 employees 已经成功创建，并且在表空间 LIN 中，该表的参数 INITIAL_EXTENT 为 100k（1021700/100 = 100k），而且 NEXT_EXTENT 也为 100k。

17.2.3　创建临时表

临时表是非常特殊的表，该表只对当前用户的当前会话有效。创建临时表的目的就是使得某些操作效率更高。临时表中的数据只对当前的会话的用户有效，是当前会话的私有数据，当前会话只操作自己的数据，没有数据锁的争用，这极大提高了临时表操作的效率。下面依次从创建临时表和临时表的可见性做详细介绍。

- 创建临时表

下面通过使用 CREATE GLOBAL TEMPORARY 指令来创建临时表。如例子 17-7 所示，该临时表为 SCOTT 用户的 EMP 表中所有 JOB 为 MANAGER 的员工信息。

例子 17-7　使用 CREATE GLOBAL TEMPORARY 指令来创建临时表

```
SQL> create global temporary TABLE
  2   scott.emp temporary
  3   on commit preserve rows
  17  as
  5   select *
  17  from scott.emp
  7   where job = 'MANAGER';

表已创建。
```



该临时表默认存储在系统的临时段中，如果临时表空间为空，也无法创建成功，会有如下错误提示。

```
SQL> create global temporary TABLE
  2  scott.emp temporary
  3  on commit preserve rows
  17  as
  5  select *
  17  from scott.emp
  7  where job = 'MANAGER';
from scott.emp
         *
ERROR 位于第 17 行:
ORA-25153: 临时表空间为空
```

遇到这样的问题，只要新建立一个临时表空间，然后改变系统的临时表空间为新建立的表空间即可。可参看第 13 章“表空间和数据文件管理”。

读者可以使用例子 17-8 所示验证是否成功创建该临时表。

例子 17-8　验证是否成功创建该临时表

```
SQL> select owner,TABLE name,TABLEspace name
  2  from dba_TABLEs
  3  where TABLE_name = 'EMP_TEMPORARY';

OWNER                TABLE NAME          TABLESPACE NAME
-------------------------------------------- ----------------------------
SCOTT                EMP_TEMPORARY
```

输出结果中 TABLESPACE_NAME 列为空，说明临时表并不是存放在默认表空间，也不存放在临时表空间中，而是存储在临时段中，临时段是一个磁盘区，当用户使用 SQL 语句执行查询时，如果需要对返回的数据进行排序时，Oracle 首先需要在内存中完成排序工作，如果内存容量不够，就需要把计算的中间结果放在临时段中。

例子 17-9　查询表 EMP_TEMPORARY 是否为临时表

```
SQL> select table name,tablespace name,temporary
  2  from dba_tables
  3  where owner = 'SCOTT'
  17  and  table name = 'EMP TEMPORARY';

TABLE NAME           TABLESPACE NAME              T
---------------------------------------------------------------------------
EMP_TEMPORARY                                       Y
```

例子 17-9 的输出说明，表 EMP_TEMPORARY 为临时表，而且该表没有存放在用户 SCOTT 的默认表空间中，而是存储在临时段中。

- 临时表的可见性

临时表在当前用户的当前会话下可用。如果用户使用其他用户登录如使用 dba 用户，或者重新启动了数据库，则无法使用该临时表。例子 17-10 说明了当使用相同的用户名，如 SYS 用户重新登录数据库时，查询临时表 emp_temporary 的输出结果。

例子 17-10　查询临时表 emp_temporary 的输出结果

```
SQL> conn /as sysdba
已连接。
SQL> desc scott.emp temporary;
 名称                                              是否为空? 类型
 ----------------------------------------------------- -------------
 EMPNO                                                     NUMBER(17)
 ENAME                                                     VARCHAR2(10)
 JOB                                                       VARCHAR2(9)
 MGR                                                       NUMBER(17)
 HIREDATE                                                  DATE
 SAL                                                       NUMBER(7,2)
 COMM                                                      NUMBER(7,2)
 DEPTNO                                                    NUMBER(2)

SQL> select *
  2  from emp_temporary;
from emp temporary
     *
ERROR 位于第 2 行:
ORA-009172: 表或视图不存在
```

例子 17-10 说明，使用 DBA（用户名为 SYS）用户重新登录数据库，此时可以查看到临时表 emp_temporary 的数据字典定义，但是不能成功查询该表中的数据。也就是说临时表只对当前用户的当前会话有效。一旦用户退出当前会话，则临时表就失去了作用。

如果不再使用临时表，则最好删除，毕竟它占用存储空间，而且一旦用户改变或重新登录，都无法重新使用该表。如例子 17-11 所示，删除临时表。

例子 17-11　删除临时表

```
SQL> DROP TABLE scott.emp temporary;

表已丢弃。
```

17.3　段空间管理

段空间管理不仅仅指表段的空间管理，其他段数据都需要自己的段管理方式，段是 Oracle 的一个逻辑结构，表段是最常用的一类段，但我们向表中插入数据、或者删除数据时以及修改数据时，都会造成段空间的变化，此时段空间的回收和分配就是一个问题。

在 Oracle 9i 之前，Oracle 使用的是手动段空间管理，此时 Oracle 要求用户自己设置诸如 FREELIST、PCTUSED、PCTFREE 等参数，通过这些参数控制端空间的使用。

在 Oracle 9i 之后开始使用自动段空间管理即 ASSM，它使用位图来管理段空间的使用情况。

如果表空间的是 ASSM，则表空间中的段自然也是 ASSM。

在 ASSM 中 FREELIST、PCTUSED 都被忽略，手工段空间管理的 MAXTRANS 参数在 11g 中被所有段忽略。

17.4 理解高水位线（HWM）

对于表段使用术语“高水位线”来标记，使用过的数据块的边界，如下所示。

```
|1|2|3|17|......|10000|未用。
```

此时第 10 000 个数据块就是这个表段的高水位线所在。

对于新建的表，高水位线位于第一个块，随着时间的推移，这些表段中存入数据，高水位线会随之移动。如果删除了表中的数据，此时某些数据块即使为空不再包含数据，但是此时依然保持高水位线的高度，但是在发生表重建、TRUNCATE 以及 SHINK 时才会调整高水位线，调整后的高水位线下的数据块都包含数据。

试想有一个超大的表，我们删除其中 90%的数据，但是高水位线没有调整，如果对该表发生了全表扫描，此时依然会对删除前的所有数据块进行扫描，依然会花费很多时间。这里是想告诉读者，高水位线 HWM 是 Oracle 访问数据实现全表扫描的数据块范围。

如果使用 TRUNCATE 而不是 DELETE 删除表中的所有数据，此时会调整高水位线到表段的第一个数据块。

ASSM（Automatic Segment Space Management，自动断空间管理）中 HWM（High Water Mark）的管理更加细致。对于 ASSM，Oracle 维护着一个低 HWM，它的含义是在低 HWM 之前的所有数据块都是有数据的，可以直接读取，而低 HWM 到 HWM 之间的所有数据需要慎重对待，这部分数据块有可能已经格式化，有可能还没有被格式化使用，此时需要参考 ASSM 的位图查看数据块使用情况。

```
|1|2|3|17|......|10000|......|15000| 使用|未用|18000
                                |                   |
                              低 HWM              HWM
```

15 000 为低 HWM，18 000 为高 HWM，在高低之间是一个“灰色“区域”，Oracle 会慎重对待，通过 ASSM 的位图查看数据块使用情况。而 15 000 之前的所有数据块不再做这种“慎重”考察，而是直接读取使用。

17.5 理解行迁移

行迁移是指某一行的数据量过大，导致该行无法存储在创建这一行的数据块中，此时 Oracle 就会使得该行离开原来的块，存储到另一个数据块中，该行的原始块和新块之间使用 ROWID 记录这种变迁关系，使得原始行知道后续数据存储的位置。

我们考虑某一行插入数据的情况，如果此时插入的数据过大，导致该行所在的数据块即使通过空闲空间合并也无法放下新插入的数据，此时 Oracle 就考虑使用行迁移，将无法放下的行数据插入新的数据块。行迁移时应该尽力避免的行为，在数据库设计时，就要根据业务数据类型，大小

设置合理的数据库块大小，或者采用其他策略从应用来避免这种行迁移的发生。

行迁移会带来许多问题。其实这个问题可以凭直觉想象，读取某一行时，本来是通过一个数据块即可读取到，此时却需要从多个数据块读取，显然增加了 I/O 时间，同时对于这种跨行存储的数据块的维护也相对复杂，多个行链势必造成维护的开销。对于有大量用户访问这种数据行时，由于多余的 I/O 以及和 I/O 相关的 LATCH（缓存多个数据块），造成该数据行的访问很明显的下降。

17.6　创建索引组织表（IOT）

索引组织表是 Oracle 提供的一种特殊类型的表，它将数据和索引存储在一起，按照索引的结构来组织和存储表中的数据。索引组织表的存储结构不是堆组织表，我们知道堆中数据的存储是无序的，而索引组织表中的数据是按照某个主键排序后存储的，然后再以 B 树的组织结构存储在数据段中。

索引组织表对于经常使用主键字段来实现查询的事务非常高效，索引组织表的主键约束不能被删除、延期或禁止。使用 IOT 表由于其索引与数据合二为一的特殊结构带来很多好处，如 IOT 节约了磁盘空间的占用，大幅度降低了 I/O 从而减少了访问缓冲区缓存，少读数据就可以不必多次读缓冲区，因为缓冲区缓存获取都需要缓冲区缓存的多个闩。

IOT 适用的场合有：

（1）完全由主键组成的表。这样的表如果采用堆组织表，则表本身完全是多余的开销，因为所有的数据全部同样也保存在索引里，此时，堆表是没用的。

（2）如果你只会通过一个主键来访问一个表，这个表就非常适合实现为 IOT。

（3）数据以某种特定的顺序物理存储，IOT 就是一种合适的结构。

IOT 提供如下的好处：

如果经常在一个主键或唯一键上使用 between 查询，也是如此。如果数据有序地物理存储，就能提升这些查询的性能。

17.6.1　IOT 表的结构

我们使用最简单的语句创建一个索引组织表，然后通过其创建元语句的查询，看看 Oracle 的索引组织表的创建到底需要什么参数，然后我们分别解释这些参数的含义以及使用场合。

下面创建一个最简单的索引组织表。

```
SQL> create table tiot
  2  (x int primary key,
  3  y number,
  17  z varchar2(20))
  5  organization index;

表已创建。
```

下面我们提取表定义元数据。

```
SQL> select DBMS_METADATA.GET_DDL('TABLE','TIOT')  FROM DUAL;

DBMS_METADATA.GET_DDL('TABLE','TIOT')
```

```
---------------------------------------------------------------------------

  CREATE TABLE "SYS"."TIOT"
   (    "X" NUMBER(*,0),
        "Y" NUMBER,
        "Z" VARCHAR2(20),
         PRIMARY KEY ("X") ENABLE
   ) ORGANIZATION INDEX NOCOMPRESS PCTFREE 10 INITRANS 2 MAXTRANS 255 LOGGING
  STORAGE(INITIAL 1755317 NEXT 101785717 MINEXTENTS 1 MAXEXTENTS 21177178317175
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "SYSTEM"
 PCTTHRESHOLD 50
```



创建 IOT 时，必须要设定主键，否则报错，索引组织表实际上将所有数据都放入了索引中。

我们通过获得表 TIOT 的定义元数据获得了 Oracle 创建一个简单的 IOT 表时的内部操作。我们需要注意与普通 HEAP 表不同的几个参数，如 ORGANIZATION INDEX、NOCOMPRESS、PCTTHRESHOLD 等。其实 IOT 表还有两个属性需要注意，这两个属性是 OVERFLOW 和 INCLUDING。

下面我们解释索引组织表属性。

- ORGANIZATION INDEX：说明这是个索引组织表。
- OVERFLOW：允许我们创建一个新段，如果 IOT 的行记录太大，则可以存储到这个新段上去。那么什么时候使用这个新段就需要考虑一下两个参数，PCTTHRESHOLD 和 INCLUDING。
- INCLUDING：行中从第一列直到 INCLUDING 所指定的列的所有列数据都存储在索引块上，其余列存储在溢出段上。
- PCTTHRESHOLD：如果行中的数据量超过了数据块大小的这个百分比，行中其余的数据就要放入溢出段了。
- COMPRESS（键压缩）：普通的索引一样，索引组织表也可以使用 COMPRESS 子句进行键压缩以消除重复值。 具体的操作是在 organization index 之后加上 COMPRESS n 子句。n 的意义在于指定压缩的列数，默认为无穷大。

17.6.2 创建 IOT 表

创建 IOT 表关键是如何使用 IOT 的参数，下面我们创建的 IOT 表的 PCTTHRESHOLD 值为 10，列 Y（包含该列）的数据溢出到 OVERFLOW 参数指定的表空间 USERS，违反 PCTTHRESHOLD 规则的行记录也会溢出到表空间 USERS。如下所示，我们创建满足该条件的 IOT 表 IOT_TEST。

```
SQL> create table iot_test
  2  (x int,
  3  y   number,
  17  z varchar2(20),
  5  constraint iot_test_pk primary key(x)
  17  )
  7  organization index
```

```
  8  pctthreshold 10
  9  including y
 10  overflow tablespace users;

表已创建。
```

通过设置合理的属性值，我们创建了一个 IOT 表，该表的主键是 X，从第二列（包含第二列）开始的所有列数据都存储到溢出段中，而对于第一列的数据只要不违反 PCTTHRESHOLD 的值就存储在索引块上，而一旦违反了这个规则就存储到溢出段中。

17.7　表参数以及参数维护

Heap Table 就是一般的表，获取表中的数据是按命中率来得到的。没有明确的先后之分，在进行全表扫描的时候，并不是先插入的数据就先获取。数据的存放也是随机的，当然根据可用空闲的空间来决定。 IOT 就是类似一个全是索引的表，表中的所有字段都放在索引上，所以就等于是约定了数据存放的时候是按照严格规定的，在数据插入以前其实就已经确定了其位置，所以不管插入的先后顺序，它在物理上的哪个位置与插入的先后顺序无关。这样在进行查询的时候就可以少访问很多 blocks，但是插入的时候，速度就比普通的表要慢一些。

在创建表时，我们使用了表空间和 storage 参数，如 initial、next、pctincrease 等，而在数据库的维护过程中，这些参数是不允许变化的，但是 Oralce 为了管理和控制数据块，而引入了 17 个参数，当使用手工管理段空间管理时，需要设置这几个参数。

- INTRANS：该参数控制对数据块的并行操作的参数。

在解释该参数前，先介绍事务槽的概念，事务槽在数据块头中，存储了有关事务的控制信息，而每行数据有一个锁位，该锁位号和事务槽号相同，数据库服务器就是通过每行的锁位找到数据块头中的事务槽，利用存储在该事务槽中的事务信息完成对该行数据的操作。每个事务只使用一个事务槽。锁位和事务槽的关系以及数据块的组成如图 17-2 所示。

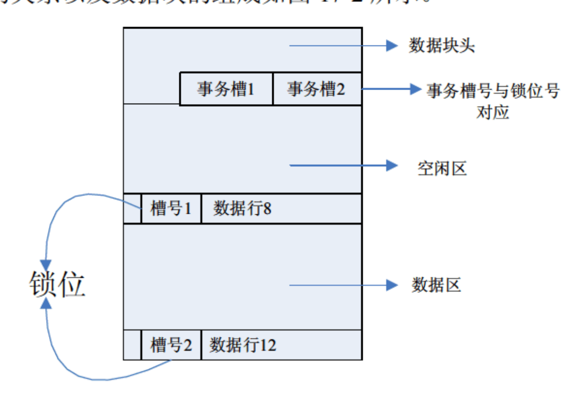


图 17-2　数据块结构及事务槽与锁位关系图

INITRANS 定义了创建数据块时事务槽的初始值，该参数的默认值为 1，如该参数为 2，说明数据库服务器实现在一个数据块中可以有最多 2 个并行的事务，每个事务独立并行地通过自己的事

务槽实现对该行数据的事务操作。

- MAXTRANS：该参数定义了创建数据块时事务槽的最大值，该参数的默认值为 255。

下面我们使用例子 17-12 来查看 SCOTT 用户的表 EMPLOYEES 的表参数 INITRANS 和 MAXTRANS。

例子 17-12　查看 SCOTT 用户的表 EMPLOYEES 的表参数 INITRANS 和 MAXTRANS

```
SQL> select ini trans,max trans
  2  from dba TABLEs
  3  where owner = 'SCOTT'
  17  and TABLE_name = 'EMPLOYEES';

 INI TRANS  MAX TRANS
-------------------- ------------------
         1          255
```

输出说明，表 EMPLOYEES 的两个参数 INITRANS 和 MAXTRANS 都采用了默认值。

- PCTFREE：该参数用于设置每个数据块中预留空间的百分比数。当数据块需要额外空间时使用 PCTFREE 参数设置的空间。该参数的默认值为 10%。

为了更好地理解 PCTFREE 参数的含义，先介绍数据块的结构，如图 17-3 所示。

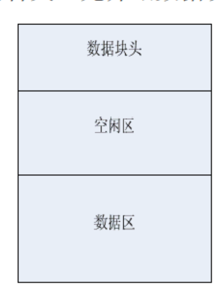


图 17-3　数据块结构图

数据块由上部分组成，即数据块头、空闲区和数据区，其中数据块头从上往下增长，而数据区是从下往上增长，但二者“碰头”则空闲区被占满。

但是一旦空闲区满，而且后续操作如修改数据增加了数据量，需要占用数据空间，此时该数据块中就无法满足要求，这样就造成占用其他数据块的空间，这种空间的置换会带来磁盘 I/O 的效率低下。所以最好在一个数据块中放置修改的数据。所以 Oracle 设计了 PCTFREE 参数，在每个数据块中设置了一个预留空间，满足在数据操作时对数据块空间的要求。

如果修改的数据行需要额外的空间，就使用 PCTFREE 参数指定的预留大小的空间。

- FREELISTS：该参数是一个空闲数据块队列的列表，当用户向表中插入数据时，需要数据块作为存储空间，那么该参数中的数据块就作为候选的数据块。
- PCTUSED：该参数定义了数据块中已经使用的空间的百分比数。只有当该数据块中已经使用的空间的百分比低于该参数值时，该数据块才放入段中的空闲数据块列表。该参数默认值为 170%。

我们先使用例子 17-13，查询 SCOTT 用户的表 EMPLOYEES（该表在 17.2 节已经创建）的参数设置。

例子 17-13　查询 SCOTT 用户的表 EMPLOYEES 的参数设置

```
SQL> select TABLE name,TABLEspace name,pct free,pct used
  2  from dba TABLEs
  3  where owner = 'SCOTT'
  17*  and TABLE_name = 'EMPLOYEES'

TABLE NAME           TABLESPACE   PCT FREE   PCT USED
---------------------------------------------------------------------------
EMPLOYEES            LIN                10         40
```

上述输出结果显示该表的参数 PCT_FREE 为 10，PCT_USED 为 40。下面使用 ALTER TABLE 指令动态修改表的参数，如例子 17-14 所示。

例子 17-14　动态修改表的参数 pctused 和 pctfree

```
SQL> ALTER TABLE scott.employees
  2  pctused 50
  3  pctfree 30;

表已更改。
```

为了验证修改结果，可以再次使用例子 17-15 来查看结果。

例子 17-15　查看例子 17-14 对表参数 pctused 和 pctfree 的修改结果

```
SQL> select TABLE_name,TABLEspace_name,pct_free,pct_used
  2  from dba TABLEs
  3  where owner = 'SCOTT'
  17  and TABLE name = 'EMPLOYEES';

TABLE_NAME           TABLESPACE   PCT_FREE   PCT_USED
-------------------- ---------- ---------- ----------
EMPLOYEES            LIN                30         50
```



由于 Oracle 中参数之间的相互关联性，修改一个参数很可能会影响其他参数的作用，所以，读者在刚开始学习 Oracle 时不要轻易修改系统的各种默认参数。

17.8　维护列

在实际的项目开发过程中，需要对建好的表进行修改，如增加或删除列修改列名等，这样的操作如果通过重新建表显然是不现实的，本节讲如何在已经建好的表中完成列维护。我们在 17.2 节已经建立了一个表 EMPLOYEES。因为该表没有数据，我们先向该表中插入数据，如例子 17-16 所示。

例子 17-16　向表 EMPLOYEES 中插入数据

```
SQL> insert into scott.employees
  2  values(1,'Tom','address1','8085173170');

已创建 1 行。
```

我们再使用例子 17-17 查询该表中的所有数据。

例子 17-17　查询表 EMPLOYEES 中的所有数据

```
SQL> select *
  2  from scott.employees;

ECODE ENAME                        EADDRESS                         EPHONE
-------------------------------------------------------------------
     1 Tom                          address1                         8085173170
```

该表有 4 列，分别为 ECODE、ENAME、EADDRESS 和 EPHONE，该表中只有一行记录，员工名字是 Tom。

（1）插入一列

下面我们演示如何插入一列，显然在表 EMPLOYEES 中没有员工性别，这显然是不合适的。我们在表中增加一列 SEX，如例子 17-18 所示。

例子 17-18　向表 EMPLOYEES 中增加一列 SEX

```
SQL>ALTER TABLE scott.employees
  2   add (
  3       sex   char
  17       );
表已更改。
```

为了验证是否增加了一列，使用例子 17-19 继续查询该表中的所有数据。

例子 17-19　验证例子 17-18 是否向表 EMPLOYEES 中增加了一列

```
SQL> col ename for a10
SQL> col eaddress for a20
SQL> col sex for a10
SQL> select *
  2* from scott.employees

ECODE ENAME      EADDRESS             EPHONE          SEX
--------------------------------------------------------------- ----------
    1 Tom        address1             8085173170
```

从输出结果看出，我们已经成功添加了一列，该列名为 SEX，但是值为空。

下面我们再增加一列并且对该列做修改，列名为 DEGREE，如例子 17-20 所示。

例子 17-20　向表 EMPLOYEES 中增加一列

```
SQL> ALTER TABLE scott.employees
  2  add (
```

```
 3     degree varchar2(10)
17      );

表已更改。
```

此时，我们更新表中的数据使得员工 Tom 的 SEX 列和 DEGREE 列都有数据，如例子 17-21 所示。

例子 17-21　更新员工 Tom 的 SEX 列和 DEGREE 列的值

```
SQL> update scott.employees
  2   set sex ='m',degree='Bachelor'
  3  where ename = 'Tom';

已更新 1 行。
```

我们查询该结果，如例子 17-22 所示。

例子 17-22　查询例子 17-21 的修改结果

```
SQL> col sex for a3
SQL> col eaddress for a10
SQL> select *
  2* from scott.employees

ECODE ENAME      EADDRESS      EPHONE           SEX     DEGREE
---------------------- --------------------------- ------ --------------
    1 Tom        address1      8085173170        m       Bachelor
```

输出显示我们成功在新添加的列中增加了数据。但是如果需要修改一个列的约束，比如不允许该列为空（NULL），则需要修改列。

（2）修改列

把列 DEGREE 设置为不允许为空（null），如例子 17-23 所示。

例子 17-23　将表 EMPLOYEES 中的列 DEGREE 设置为不允许为空

```
SQL> ALTER TABLE scott.employees
  2  modify(
  3  degree varchar2(10) not null
 17       )
  5  ;

表已更改。
SQL> desc scott.employees;
 名称                                   是否为空? 类型
 ----------------------------------------- -------- ----------------
 ECODE                                           NUMBER(17)
 ENAME                                           VARCHAR2(25)
 EADDRESS                                        VARCHAR2(30)
 EPHONE                                          VARCHAR2(15)
 SEX                                             CHAR(1)
 DEGREE                                 NOT NULL VARCHAR2(10)
```

我们成功修改了列 DEGREE 的约束，不允许该列为空，如果用户再次插入数据时，该列为空则无法成功插入。

（3）删除列

用户既然可以修改表中的列，添加列。自然也可以删除不需要的列。删除列的语法格式为：

```
ALTER TABLE tablename DROP COLUMN columnname CASCADE CONTRAINTS
```

参数 CASCADE CONSTRAINTS 不是必需的，但是如果该列是一个表的外键，也就是说该表是一个外键引用的父表，而该外健就是此时要删除的列，则需要使用 CASCADE CONSTRAINTS 参数。

该操作对于 Oracle 8i 以上的版本都适用，但是使用该指令时，数据库系统会重新将表写入磁盘，目的是为了还原需要，这样对于一个大表就会占用较大的还原空间，一旦删除该列就无法恢复。

例子 17-24　删除表 EMPLOYEES 中的列 DEGREE

```
SQL> ALTER TABLE scott.employees DROP COLUMN degree;

表已更改。
SQL> desc scott.employees;
 名称                                      是否为空? 类型
 ----------------------------------------- -------- -------------
 ECODE                                              NUMBER(17)
 ENAME                                              VARCHAR2(25)
 EADDRESS                                           VARCHAR2(30)
 EPHONE                                             VARCHAR2(15)
 SEX                                                CHAR(1)
```

上述输出说明，成功删除表 EMPLOYEES 中的列 DEGREE。

在很大的表中删除一行非常耗费时间，此时可以在 ALTER TABLE 语句中使用 SET UNUSED 子句，这样就将表中某列置成无用的列。其语法如下：

```
ALTER TABLE <username.>tablename
SET UNUSED COLUMN columnname CASCADE CONSTRAINTS;
```

其实，此时并没有删除表中的该列的数据，而是使得用户查询时看不到该列内容，该列被数据库认为是删除的列。并且一旦该列设置为 UNUSED 则无法恢复。但数据库空闲时，可以使用如下命令再删除置为无用的列。

```
ALTER TABLE <username.>tablename
DROP UNUSED COLUMNS ;
```

下面演示如何将一列设置成无用的列。如例子 17-25 所示，将 EPHONE 列设置为无用的列。

例子 17-25　将表 EMPLOYEES 中的 EPHONE 列设置为无用的列

```
SQL> ALTER TABLE scott.employees
  2* SET UNUSED  COLUMN ephone

表已更改。
SQL> select *
  2  from scott.employees;
```

```
ECODE ENAME              EADDRESS            SEX
---------- -------------------- -------------------- ---
         1 Tom           address1            m
```

表 EMPLOYEES 中的列 EPHONE 设置为无用，此时数据库认为该列已经删除，在使用 SQL 语句查询时会发现没有列 EPHONE 的值。

下面删除掉设置为 UNUSED 的列。

例子 17-26　删除掉表 EMPLOYEES 中设置为 UNUSED 的列

```
SQL> alter table scott.employees
  2  drop unused columns;

表已更改。
```

（4）更改列名字

列名是程序员设计用来表示一个字段的信息，如果在开发过程中由于某种原因需要对列名进行规范，则需要对已经创建好的表的列名进行修改。修改列名的语句很简单，如下所示。

```
ALTER TABLE <username.>employees
RENAME COLUMN old_columnname
TO new_columnname;
```

例子 17-27 给出修改列名的实例演示，该实例将表 EMPLOYEES 中的列 SAL 改为 SALARY。

例子 17-27　将表 EMPLOYEES 中的列 SAL 改为 SALARY

```
SQL> alter table scott.emp
  2  rename column sal
  3  to salary;

表已更改。
```

我们查询更改结果，如下所示。

```
SQL> desc scott.emp;
 名称                                      是否为空? 类型
 ----------------------------------------- -------- ------------------------
 EMPNO                                     NOT NULL NUMBER(17)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(9)
 MGR                                                NUMBER(17)
 HIREDATE                                           DATE
 SALARY                                             NUMBER(7,2)
 COMM                                               NUMBER(7,2)
 DEPTNO                                             NUMBER(2)
```

我们在输出中可以看到原表中的 SAL 列名已经改为 SALARY 了。说明例子 17-27 修改列名成功。

17.9 删除和截断表

在不需要一个表时，可以删除表，使用 DROP TABLE 语句，此时会彻底删除表中的数据和表的结构，表的结构指在创建表时定义的列名、列属性和一些约束信息。

如果只想删除表中的数据可以使用 TRUNCATE TABLE 语句截断一个表，此时会保留表的结构，只删除表中的数据。本节讲解和演示如何删除和截断表。

为了演示如何删除或截断一个表，先使用例子 17-28 创建一个表 EMP_TEMP。

例子 17-28　创建一个表 EMP_TEMP

```
SQL> create TABLE scott.emp temp
  2  as
  3  select *
  17  from scott.emp;

表已创建。
```

（1）截断表（TRUNCATE）

不严谨地讲，表由表结构和表中的数据组成，如果不需要表中的数据，可以使用 TRUNCATE 来截断一个表。使用该指令截断表时，只删除表中的数据但是保留表的结构，也就是对表的定义还是存在的，可以使用 INSERT 指令向表中添加数据。

该指令的语法格式为：

```
TRUNCATE TABLE username.TABLEname。
```

为了使得读者便于直观理解，这里给出一个例子，如例子 17-29 所示，然后再介绍 TRUNCATE 的其他特性。

例子 17-29　截断表 emp_temp

```
SQL> truncate TABLE scott.emp_temp;

表已截掉。
```

输出说明用户 SCOTT 中的表 emp_temp 已经截断，我们再使用例子 17-30 验证改变表的结构，也就是对表的定义是否还存在。

例子 17-30　验证表 emp_temp 的定义是否还存在

```
SQL> desc scott.emp_temp;
 名称                                              是否为空? 类型
 ------------------------------------------------- ------------ -------------
 EMPNO                                                         NUMBER(17)
 ENAME                                                         VARCHAR2(10)
 JOB                                                            VARCHAR2(9)
 MGR                                                           NUMBER(17)
 HIREDATE                                                       DATE
 SAL                                                            NUMBER(7,2)
 COMM                                                          NUMBER(7,2)
 DEPTNO                                                         NUMBER(2)
```

显然，输出结果说明用户 SCOTT 的表 emp_temp 依然存在。我们再使用例子 17-31 来验证表中是否还有数据。

例子 17-31　验证表 emp_temp 中是否还有数据

```
SQL> select *
  2  from scott.emp temp;

未选定行
```

显然，用户 SCOTT 的表 emp_temp 中没有数据。

下面我们总结一下 TRUNCATE 指令的特性，这些特性来源于 ORACLE 的文档。

- 删除表中的数据但是保留了表结构。
- 数据一旦删除就释放数据占有的磁盘空间，并且数据不可恢复。
- 如果表被外键引用，则无法使用 TRUNCATE 删除表中的数据。
- 所有和表相关的索引也被截断。
- 不会触发删除表的删除触发器。

在读者学习了索引和触发器后可以使用例子来验证上述对于 TRUNCATE 指令特性的说明是否正确。这里就不再给出具体示例。

（2）删除表

如果不再需要表，包括表中的数据和表的结构，则使用 DROP 指令。使用该指令删除表的结构，表中的数据无法恢复，所以在使用该指令前务必要确认不再需要该表。使用 DROP 删除表的语法格式为：

```
DROP TABLE [username.]TABLEname
CASCADE CONSTRAINTS;
```

当一个表被 DROP 后，该表使用的 EXTENTS 得到释放，如果这些 EXTENTS 是连续分配的，则这些连续区域得到释放并组合成更大的可分配空间。如果该表的外键被另一个表引用，即该表在外键关系中是父表，则需要使用 CASCADE CONSTRAINTS。使得该表脱离与子表的关系，并顺利删除。在维护数据完整性时会再次讲到 CASCADE CONSTRAINTS。读者可以暂时不关系该参数的使用。

为了说明删除一个表，我们使用例子 17-32 来说明，在使用 TRUNCATE 删除用户 SCOTT 的表 emp_temp 时，正是删除了表中的数据，下面就使用 DROP 指令删除表结构。

例子 17-32　使用 DROP 指令删除表 emp_temp

```
SQL> DROP TABLE scott.emp_temp;

表已丢弃。
```

我再使用例子 17-33，验证该表的结构和表中数据是否存在。

例子 17-33　验证 SCOTT 用户的表 emp_temp 结构是否存在

```
SQL> desc scott.emp_temp;
```

```
ERROR:
ORA-0170173: 对象 scott.emp_temp 不存在
```

显然用户 SCOTT 的表 emp_temp 已经删除，因为错误提示为“对象 scott.emp_temp 不存在”。当然也无法查询该表中的数据。

17.10　本章小结

本章重点介绍了如何管理和维护表。数据类型是在表维护中非常重要的概念。本章介绍了如何创建普通表以及临时表。表的管理部分介绍了段空间管理方式，因为不同方式的选择会影响表的存取效率，高水位线、行迁移也是需要重点理解内容，最后介绍了索引组织表，使得读者理解 IOT 的使用场合、优缺点以及如何创建满足需要的 IOT 表。

第 18 章
◀索　引▶

索引是 Orcale 实现数据访问非常重要和常用的方法，本章将重点介绍索引的原理以及各类索引的使用，还将介绍索引的扫描类型、限制索引使用的条件，以及 Oracle 支持的索引监控和索引维护。集群因子、二元高度、直方图都是索引使用中要注意的问题。

18.1　索引的概念

索引是 Oracle 的一个对象，是否使用索引由 Oracle 决定，索引中存储了特定列的排序数据，实现对表的快速访问。使用索引可以很快查找到建立索引时列的值所在的行，而不必对表实现全表扫描，所以适当地使用索引可以减少磁盘 I/O 量。在开始我们给出索引的特点总结，这样读者在接下来使用索引时，脑子中就有一个局限，不要认为使用索引就是好事☺。

索引的特点：

- 对于具有只读特性或较少插入、更新或删除操作的大表通常可以提高查询速度。
- 可以对表的一列或多列建立索引。
- 建立索引的数量没有限制。
- 索引需要磁盘存储，需要 Oracle 自动维护。
- 索引对用户透明，是否使用索引是 Oracle 决定的。

读者都有这样的体验，如果去图书馆借书，首先会到查询机前检索需要的书籍，然后确认该书的具体位置，通常该位置确定该书在图书室的哪个区域的哪个书架。这样只要径直到具体的地点去找这本书，很快就可以找到。但是试想，如果你面对国家图书馆新馆的阅览室，四处都是高大的书架，那种茫然是可以想象的。Oracle 使用索引的目的也是为了迅速地找到需要的数据，当然任何事物都有两面，建立索引可以迅速找到需要的数据，但是在某些情况下也带来性能开销，我们在本章将分析索引的建立和使用维护，并给出索引使用的一些建议。

18.2　Oracle实现数据访问的方法

Oracle 的 RDBMS 在访问数据时使用最根本的 3 种访问方法：

- 全表扫描；

- 通过 ROWID;
- 使用索引。

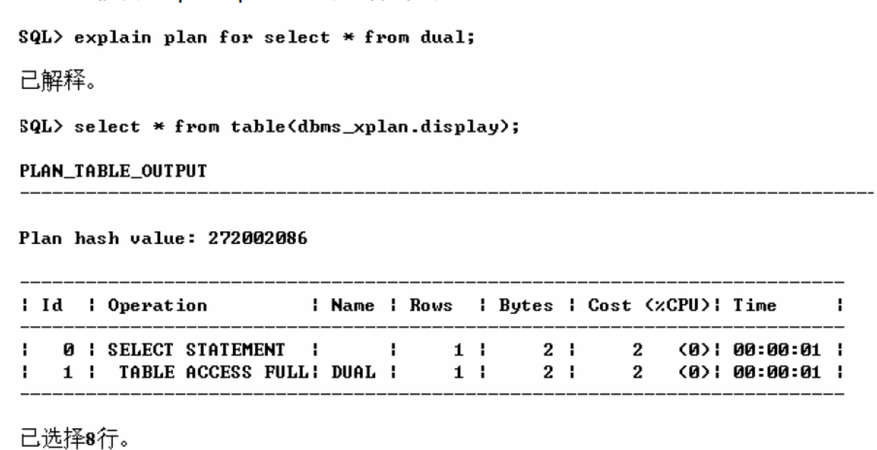
当 Oracle 决定使用索引时会使用 ROWID 来访问数据，当没有索引或者不选择使用索引时就使用全表扫描的方式。Oracle 中扫描数据（数据访问路径）的方法。

18.2.1 全表扫描（FULL TABLE SCAN)

在使用全表扫描时，Oracle 读取表中所有的行，此时通过多块读操作可以大大减少 IO 的次数，利用多块读可以大大提高全表扫描的速度，只有在全表扫描的情况下才能使用多块读。在较大的表上不建议使用全表扫描，如果读取表的数据总量超过 5%~10%，那么通常进行全表扫描。并行查询可能会使得我们的路径选择采用全表扫描。

即使在表上创建了索引，是否使用该索引也由 Oracle 根据 CBO 优化器计算的结果选择，用户无法干预（当然 DBA 可以修改参数或者 SQL 语句）。

例子 18-1　使用 explian plan 解释执行计划



```
SQL> explain plan for select * from dual;

已解释。

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

Plan hash value: 272002086

--------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |     2 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| DUAL |     1 |     2 |     2   (0)| 00:00:01 |
--------------------------------------------------------------------------

已选择8行。
```

18.2.2　通过行 ID（ROWID）

对于表对象，在向表中插入数据时隐含会创建该行的 ROWID，ROWID 是数据行所存储的数据块地址，这样就以最快的速度找到该行数据。如例子 8-2 所示，查询行 ID，我们通过比较看全表扫描和通过行 ID 哪个更快。

ROWID 指出了数据文件、块号、行号，通过 ROWID 是 Oracle 数据库中读取单行数据最快速的方法。这种方法不会采用多块读，而是会采用单块读的方式。

例子 18-2　通过执行计划确定单块读

```
SQL> explain plan for select * from dept where rowid = 'AAAAyGAADAAAAATAAF';
已解释。
SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 3453257278

--------------------------------------------------------------------------------
---
| Id  | Operation                   | Name | Rows  | Bytes | Cost (%CPU)| Time
  |
--------------------------------------------------------------------------------
---

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |      |     1 |    20 |     1   (0)| 00:00:0
1 |
|   1 |  TABLE ACCESS BY USER ROWID| DEPT |     1 |    20 |     1   (0)| 00:00:0
1 |
--------------------------------------------------------------------------------
---
已选择8行。
```

18.2.3　使用索引

通过索引找到数据行的 ROWID、然后通过 ROWID 直接到表中查找数据，这种方式称为索引查找或者索引扫描。因为一个 ROWID 对应一个数据行，因此这种方式采用的也是单块读。在索引中，除了存储每个索引值，还存储相应的 ROWID，索引扫描分为两步：

（1）扫描索引得到相应的 ROWID。

（2）通过找到的 ROWID 从表中读取相应的数据。

每次采用的都是单块 I/O 读，因为索引小而且经常使用，因此通常被 cache 到内存中，因此第一步通常是逻辑读（数据可以从内存中得到），因为表数据比较大，因此第二步读通常是物理读，因此性能较低。

索引改进性能的程度取决于两个因素：

- 数据的选择性;
- 表数据在数据块上的分布。

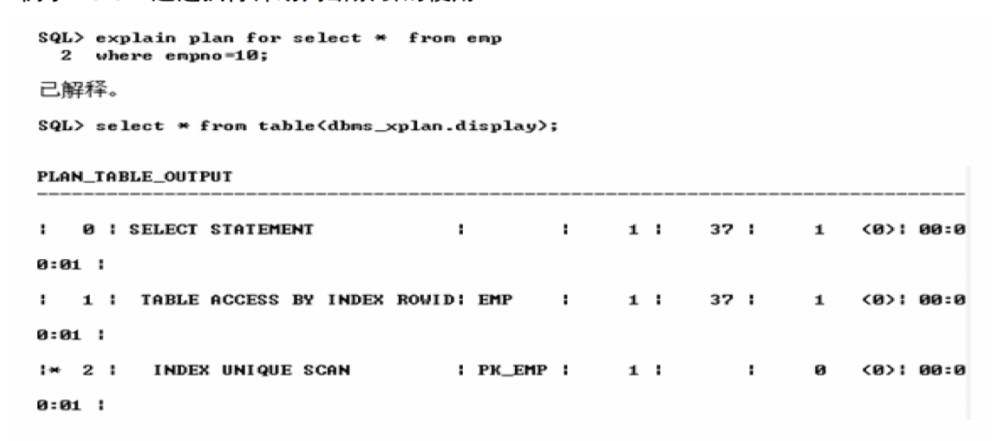
如果选择性很高（例如身份证号码），那么根据索引值返回的 ROWID 很少，如果选择性很低（例如国家）则返回的 ROWID 很多，那么索引的性能将会大大降低（返回的数据少，I/O 压力小）。

如果选择性很高，但是相关的行在表中的存储位置并不互相靠近，则会进一步减少索引的益处，如果匹配索引值的数据分散在表的多个数据块中，则必须从表中选择多个单独的块以满足查询，基于索引的读取是单块读取，如果使用全表扫描，使用的是多块读取以快速扫描表，因此全表扫描不见得比索引扫描速度慢。关键看数据对象数据块的分散程度。

当访问的行数较少时，SELECT、UPDATE、DELETE+WHERE 条件可以从索引中得到更多的好处。一般来说，增加索引会带来 insert 语句性能的下降，如果根据未索引列 update 索引列，那么

也会带来性能的降低。大量的 delete 也会因为索引的存在而导致性能降低。因此我们要分析具体的情况，判断索引和 DML 语句之间的关系。

例子 18-3　通过执行计划判断索引的使用



```
SQL> explain plan for select *  from emp
  2  where empno=10;
已解释。
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |        |     1 |    37 |     1   (0)| 00:0
0:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |     1 |    37 |     1   (0)| 00:0
0:01 |
|*  2 |   INDEX UNIQUE SCAN         | PK_EMP |     1 |       |     0   (0)| 00:0
0:01 |
```

访问路径走的是主键索引，因此是 INDEX UNIQUE SCAN，首先是索引扫描、然后是根据索引查找到的 ROWID 进行表的访问。

18.3　索引扫描类型

Oracle 在使用索引扫描数据时，根据索引的类型、数据分布以及 SQL 的谓词等确定索引的扫描类型。索引扫描的 4 种类型。

- 索引唯一扫描（INDEX UNIQUE SCAN);
- 索引范围扫描（INDEX RANGE SCAN);
- 索引全扫描（INDEX FULL SCAN);
- 索引快速扫描（INDEX FAST FULL SCAN)。

下面分别通过例子介绍什么条件下发生此种类型的索引扫描。

18.3.1　索引唯一扫描（INDEX UNIQUE SCAN)

索引唯一扫描指通过唯一键、主键，Oracle 通常返回一个数据行，因此采用的是索引唯一扫描。

例子 18-4　索引唯一扫描

```
SQL> explain plan for select *  from emp
  2  where empno=10;
已解释。
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |        |     1 |    37 |     1   (0)| 00:0
0:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |     1 |    37 |     1   (0)| 00:0
0:01 |
|*  2 |   INDEX UNIQUE SCAN         | PK_EMP |     1 |       |     0   (0)| 00:0
0:01 |
```

18.3.2　索引范围扫描 (INDEX RANGE SCAN)

当发生如下情况时 Oracle 将使用索引范围扫描。

- 在唯一键上使用 range 操作符（>、<、<>、>=、<=、between）;
- 在组合索引上，只使用部分列进行查询，导致查询出多行;
- 对非唯一索引列上进行的查询。

例子 18-5　索引范围扫描

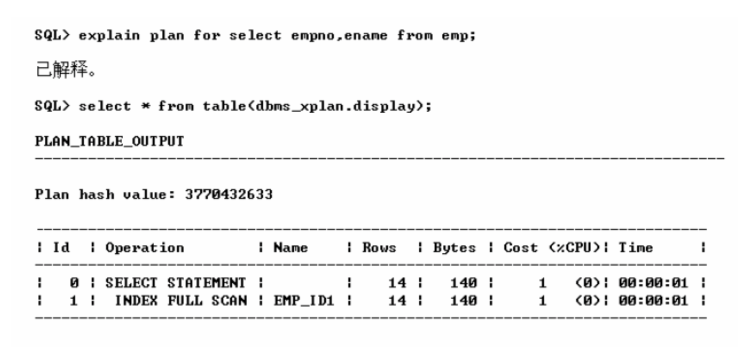
```
SQL> explain plan for select * from emp where empno>10;
已解释。
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |    14 |   518 |     2   (0)| 00:
00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP     |    14 |   518 |     2   (0)| 00:
00:01 |
|*  2 |   INDEX RANGE SCAN          | EMP_ID1 |    14 |       |     1   (0)| 00:
00:01 |
```

18.3.3　索引全扫描 (INDEX FULL SCAN)

当使用索引全扫描时，查询出的数据必须全部从索引中得到，如例子 18-6 所示。

例子 18-6　索引全扫描



```
SQL> explain plan for select empno,ename from emp;
已解释。
SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

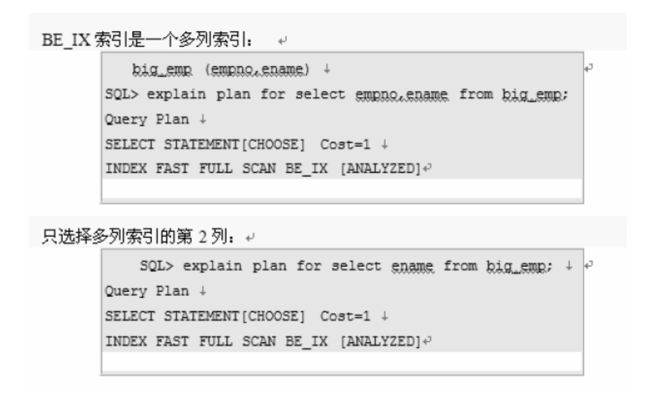
Plan hash value: 3770432633

---------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |    14 |   140 |     1   (0)| 00:00:01 |
|   1 |  INDEX FULL SCAN  | EMP_ID1 |    14 |   140 |     1   (0)| 00:00:01 |
---------------------------------------------------------------------------
```

18.3.4　索引快速扫描（INDEX FAST FULL SCAN)

索引快速扫描扫描索引块中的所有数据块，这点与 INDEX FULL SCAN 相似，但是索引快速扫描不进行数据的排序，在这种方式下，可以使用多块读功能、也可以使用并行读功能，最大化数据的吞吐量。

例子 18-7　索引快速扫描



BE_IX 索引是一个多列索引：

```
   big_emp  (empno,ename)
SQL> explain plan for select empno,ename from big_emp;
Query Plan
SELECT STATEMENT[CHOOSE]  Cost=1
INDEX FAST  FULL  SCAN BE_IX  [ANALYZED]
```

只选择多列索引的第 2 列：

```
    SQL> explain plan for select ename from big_emp;
Query Plan
SELECT STATEMENT[CHOOSE]  Cost=1
INDEX FAST  FULL  SCAN BE_IX  [ANALYZED]
```

18.4　限制索引使用的情况

限制索引的使用，很多情况下即使创建了索引，也会导致索引不能使用，如使用了函数而没有创建基于该函数的索引。

我们下面来研究一下使用 where 但是阻止 Oracle 使用索引的几种情况。

18.4.1　使用不等于运算符

使用不等于运算符（<>、!=），在 where 中使用这些不等于条件，将会使索引失效。

例子 18-8　不等运算使得索引失效

```
SQL> explain plan for select * from emp where empno<>10;

已解释。

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

Plan hash value: 3956160932

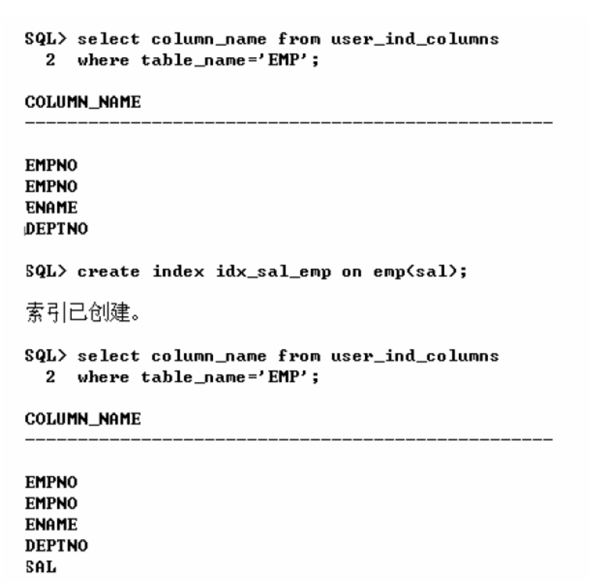
--------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |    14 |   448 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |    14 |   448 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------
```

当分析表的时候，Oracle 收集表中数据分布的相关统计信息，通过使用这种分析，基于成本的优化器可以决定在 where 子句中对一些值使用索引，而对其他的值不使用索引。因此不是说在一个列上建立了索引就一直会使用索引。根据不同值，优化器都会确定是否使用索引。

18.4.2　使用 IS NULL 或 IS NOT NULL

在 where 子句中使用 IS NULL 或者 IS NOT NULL 同样会限制索引的使用。如果被索引的列在某些行中存在 NULL 值，在索引列中就不会有相应的条目。位图索引对于 NULL 列也会进行记录，因此位图索引对于 NULL 搜索通常较为快速。

例子 18-9　NULL 判断影响索引使用



```
SQL> select column_name from user_ind_columns
  2  where table_name='EMP';

COLUMN_NAME
--------------------------------------------------

EMPNO
EMPNO
ENAME
DEPTNO

SQL> create index idx_sal_emp on emp(sal);

索引已创建。

SQL> select column_name from user_ind_columns
  2  where table_name='EMP';

COLUMN_NAME
--------------------------------------------------

EMPNO
EMPNO
ENAME
DEPTNO
SAL
```

在例子 18-9 中，我们基于表 EMP 的 SAL 列创建了 B 树索引。下面在查询谓词中使用 is NULL 来查询。

例子 18-10　不走索引走全表扫描

```
SQL> explain plan for select empno,ename,deptno
  2  from emp
  3  where sal is null;

已解释。

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
----------------------------------------------------------------------------

Plan hash value: 3956160932

--------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    13 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    13 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------
```

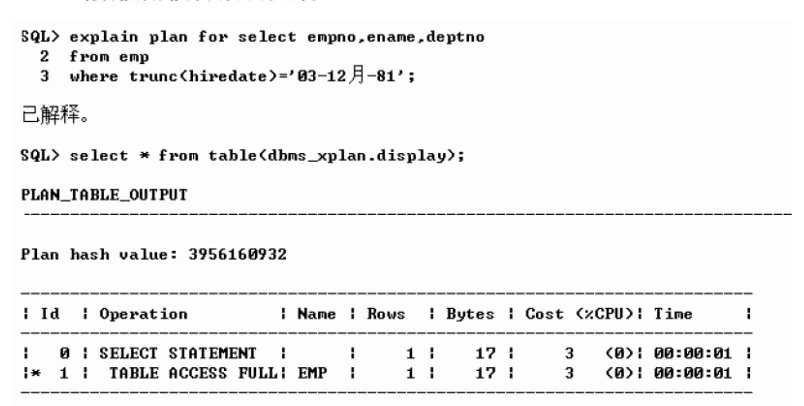
此时使用了全表扫描，因此建议对列加上 NOT NULL 或者 DEFAULT ，防止 NULL 值的出现，从而导致该列上的索引不能使用。

18.4.3　使用函数

如果不使用基于函数的索引（后面会讲到），那么在 SQL 语句的 where 子句中对存在索引的列使用函数时，会使优化器忽略掉这些索引。

一些常见的函数如 trunc、substr、to_date、to_char、instr 等，都可能会使索引失效。

例子 18-11　函数使用使得索引失效



```
SQL> explain plan for select empno,ename,deptno
  2  from emp
  3  where trunc(hiredate)='03-12月-81';

已解释。

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------

Plan hash value: 3956160932

--------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    17 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    17 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------
```

分析函数的使用复制了索引的使用，此时 SQL 查询走全表扫描。

解决方案：

（1）使用基于函数的索引。

（2）灵活书写 SQL，避免在索引列上使用 SQL 函数。

例子 18-12　修改 SQL 防止函数对索引的影响。

```
SQL> explain plan for select empno,ename,deptno
  2  from emp
  3  where hiredate>'03-12月-81'
  4  and hiredate < (to_date('03-12月-81')+0.99999);

已解释。

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

|   0 | SELECT STATEMENT            |             |     1 |    17 |     2  (0
)| 00:00:01 |

|*  1 |  FILTER                     |             |       |       |
 |          |

|   2 |   TABLE ACCESS BY INDEX ROWID| EMP        |     1 |    17 |     2  (0
)| 00:00:01 |

|*  3 |    INDEX RANGE SCAN          | IDX_HIRE_EMP |     1 |       |     1  (0
)| 00:00:01 |
```

此时，通过灵活一变，SQL 语句的访问路径变成了走索引。为了走索引，就不要在 where 子句中的索引列上使用函数。

18.4.4　比较不匹配的数据类型

这个是比较难于发现的问题。Oracle 不会对不匹配的数据类型报错，Oracle 会隐式地把 VARCHAR2 列的数据类型转换成要被比较的数值型数据类型（这是一个例子，还存在其他的数据类型转换）。

例子 18-13 隐式转换影响索引使用。

```
SQL> explain plan for select bank_name,address,city,state,zip
  2  from banks
  3  where zip=100043;

已解释。

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

Plan hash value: 1103983463

---------------------------------------------------------------------------
| Id  | Operation         | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |     1 |    69 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| BANKS |     1 |    69 |     2   (0)| 00:00:01 |
---------------------------------------------------------------------------
```

这里自动加了一个函数 to_number(zip)，对于不匹配的数据类型，Oracle 隐式地加上一个转换函数。为什么不加载到 100043 上面，因为这是一个常量，常量是不能改变的。

解决方案是灵活地使用，在常量上面加上一个’ ’，表示这是一个字符常量，这样字符常量就和 ZIP VARCHAR2 一致了。索引列的数据类型和常量类型要求一致。

例子 18-14 使用字符常量解决隐式转换。

```
SQL> explain plan for select bank_name,address,city,state,zip
  2  from banks
  3  where zip='100043';
```

18.5 集群因子

优化器决定是否使用索引的两个关键因素。

- 选择性：选择性高意味着不同的值越多，使用索引时返回的数据量越少。
- 集群因子：集群因子是索引与他所基于的表相比较得出的有序性度量，它用于检查在索引访问之后执行的表查找的成本。使用索引访问数据之后的表查找成本计算依据。

什么是集群因子呢？集群因子的含义是如果通过一个索引扫描一张表，需要访问表的数据块的数量。集群因子计算的方法如下：

（1）扫描一个索引。

（2）比较某行的 ROWID 和前一行的 ROWID，如果这两个 ROWID 不属于同一个数据块，那么集群因子增加 1。

（3）整个索引扫描完毕后，就得到了该索引的集群因子。

如果集群因子接近于表存储的块数，说明这张表是按照索引字段的顺序存储的。如果集群因子接近于行的数量，那说明这张表不是按索引字段顺序存储的。在计算索引访问成本的时候，这个值十分有用。集群因子乘以选择性参数（selectivity）就是访问索引的开销。

如果这个统计数据不能真实反映出索引的真实情况，那么可能会造成优化器错误的选择执行计划。另外如果某张表上的大多数访问是按照某个索引做索引扫描，那么将该表的数据按照索引字段的顺序重新组织，可以提高该表的访问性能。

18.6 二元高度

索引查找分为两个过程：

（1）根据树进行定位、找出 ROWID（索引查找）。

（2）根据 ROWID 找出表中的数据行（表数据查找）。

进行索引查找的时候，首先从树根开始读数据，通过中间节点，最后定位到叶节点，整个过程只能进行单数据块的读取，如图 18-1 所示。

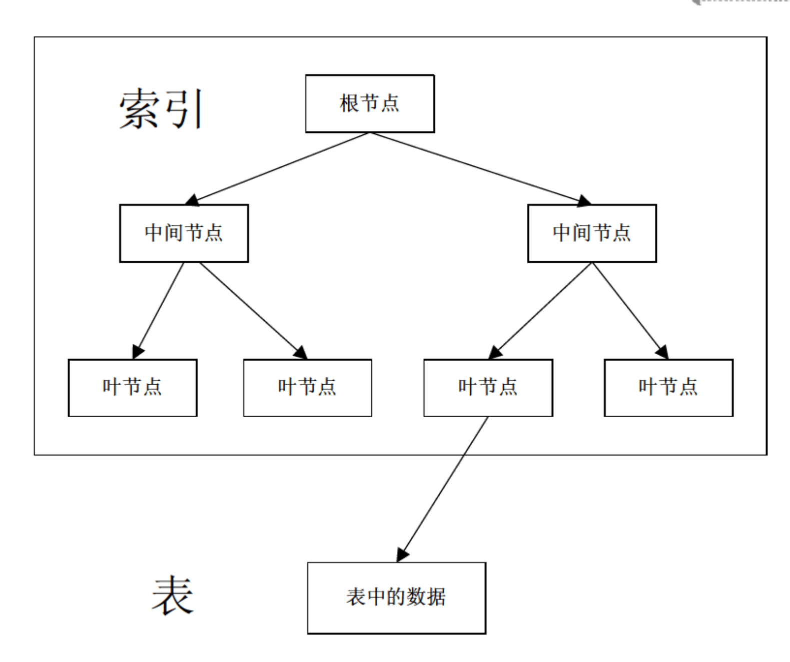
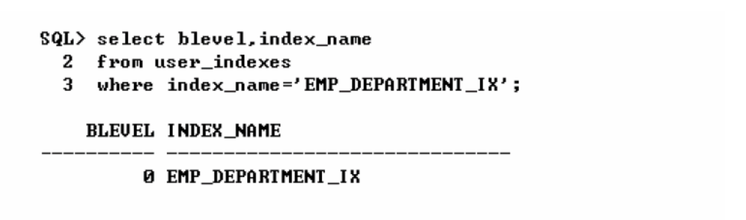


图 18-1 索引查询数据示意图

在图 18-1 中，索引的二元高度是 3，读取一行数据经过了四次数据块的读取，其中三次是索引，一次是表数据。

例子 18-15　查询索引的二元高度。



```
SQL> select blevel,index_name
  2  from user_indexes
  3  where index_name='EMP_DEPARTMENT_IX';

    BLEVEL INDEX_NAME
---------- ------------------------------
         0 EMP_DEPARTMENT_IX
```

查找每个索引的二元高度，这个索引只占用了一个数据块，因此二元高度是 0。

二元高度主要随着表中索引列的非 NULL 值以及索引列中值的宽度而变化。如果索引列上大量的行被删除，那么他的二元高度不会降低，重建索引会降低二元高度，如果一个索引中被删除的行接近 20～25%，重建索引会减低二元高度。二元高度对索引的性能影响不是很大，但是在可能的情况下，降低二元高度还是有必要的。

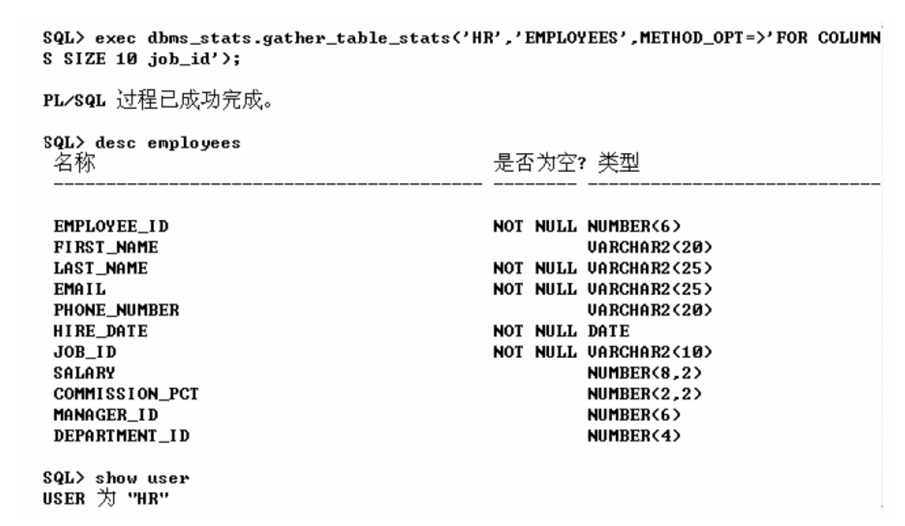
18.7　直方图

在分析表和索引时，直方图用于记录数据的分布。通过获取该信息，基于成本的优化器就可以决定使用将返回少量行的索引，而避免使用基于限制条件返回许多行的索引。直方图的使用不受索引的限制，我们可以在表的任何列上构建直方图（一般是在表的索引列上构建直方图）。

构建直方图最主要的原因就是：帮助优化器在表中数据严重倾斜时做出更好的规划。如果一

个表中的列上（通常是索引列）数据发生严重的倾斜，那么在这个列上建立直方图将非常地有意义。这样优化器就知道什么时候该使用索引，什么时候不该使用索引了。

例子 18-16　收集表 employees 包含直方图的统计信息



```
SQL> exec dbms_stats.gather_table_stats('HR','EMPLOYEES',METHOD_OPT=>'FOR COLUMN
S SIZE 10 job_id');

PL/SQL 过程已成功完成。

SQL> desc employees
 名称                                        是否为空? 类型
 ----------------------------------------- -------- ----------------------------

 EMPLOYEE_ID                               NOT NULL NUMBER(6)
 FIRST_NAME                                         VARCHAR2(20)
 LAST_NAME                                 NOT NULL VARCHAR2(25)
 EMAIL                                     NOT NULL VARCHAR2(25)
 PHONE_NUMBER                                       VARCHAR2(20)
 HIRE_DATE                                 NOT NULL DATE
 JOB_ID                                    NOT NULL VARCHAR2(10)
 SALARY                                             NUMBER(8,2)
 COMMISSION_PCT                                     NUMBER(2,2)
 MANAGER_ID                                         NUMBER(6)
 DEPARTMENT_ID                                      NUMBER(4)

SQL> show user
USER 为 "HR"
```

在 hr 用户下面的 employees 表的 job_id 列上建立了一个直方图。这个直方图有 10 个存储桶，如图 18-2 所示。

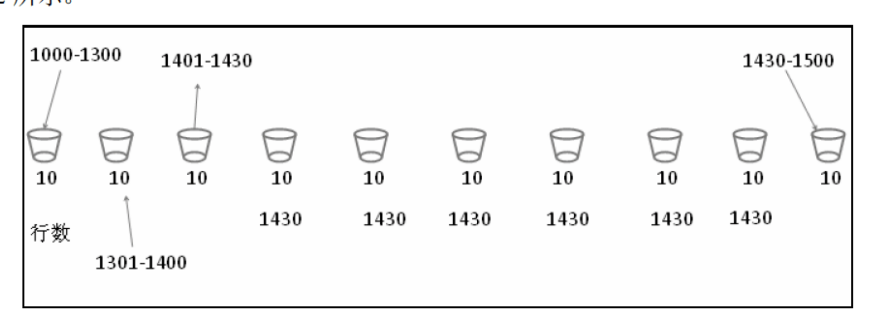


图 18-2　直方图

将整个列的 100 行数据均分成 10 个存储桶，每个桶中存储 10 个数据行。然后写出每个桶中的数据范围。我们发现 1430 这个数值对应的行号数非常多，大约 73 行，占到了 73%。因此当我们使用 where id=1403 的时候，优化器不会走索引。

默认情况下，Oracle 的直方图会产生 75 个存储桶，可以把存储桶的 size 限制在 1～254 之间。

18.8　建立索引

Oracle 建立索引使用 CREATE INDEX 指令，我们通过例子 18-17 说明如何建立索引，此时我们使用 SCOTT 用户的表 EMP 作为例子，我们假设该表是很大的表，且用户经常使用用户名查询该表中的数据。

例子 18-17　对 EMP 表建立索引

```
SQL> conn scott/oracle
已连接。
SQL> create index emp_ename_idx
  2  on emp(ename);

索引已创建。
```

此时，对表 EMP 的列 ENAME 建立了索引，我们知道索引是需要存储空间的，也就是索引也占用磁盘空间的。那么索引存储在哪个表空间，以及如何查看已经建立的索引的信息呢？Oracle 使用 USER_INDEXES 数据字典实现，如例子 18-18 所示。

例子 18-18　使用 USER_INDEXES 数据字典查询索引信息

```
SQL> col index_name for a20
SQL> col index_type for a10
SQL> col table_name for a20
SQL> col tablespace_name for a20
SQL> select index_name,index_type,table_name,tablespace_name
  2* from user_indexes

INDEX_NAME       INDEX_TYPE TABLE_NAME       TABLESPACE_NAME
----------------------------                  --------------------
------------------------------- ------------------------------
PK_EMP               NORMAL     EMP              USERS
EMP_ENAME_IDX        NORMAL     EMP              USERS
PK_DEPT              NORMAL     DEPT             USERS
```

使用数据字典 USER_INDEXES 可以详细地查看当前用户所拥有的索引信息，如上输出所示，我们刚刚建立的索引名字 INDEX_NAME 为 EMP_ENAME_IDX，该索引所依赖的表 TABLE_NAME 为 EMP，存储在用户表空间 USERS 中。这样通过数据字典视图 USER_INDEXES 可以很清楚地知道关于当前的所有索引信息。

读者在索引维护中需要知道索引所对应的表空间的信息，因为需要了解该表空间所在的磁盘 I/O 或该磁盘的使用情况来平衡磁盘读写。

例子 18-19　查看索引所对应的表空间信息

```
SQL> col "索引名" for a118
SQL> col "索引对应的表空间名" for a20
SQL> col "索引对应的磁盘文件" for a40
SQL> select a.index_name "索引名" ,a.tablespace_name "索引对应的表空间名
" ,b.file_name "索引对应的磁盘文件"
  2  from dba_indexes a,dba_data_files b
  3  where a.index_name like 'EMP%'
  4* and a.tablespace_name = b.tablespace_name

索引名              索引对应的表空间名   索引对应的磁盘文件
-------------------------------- ----------------------------------------
EMP_ENAME_IDX    USERS                 F:\ORACLE\PRODUCT\10.2.0\ORADATA\LEEJIA\
                                       USERS01.DBF
```

下面我们再创建一个索引，在表 EMP 的列 ENAME 和 SAL 上创建多列索引，并且指定表空间。首先创建一个索引表空间。

例子 18-20　创建索引表空间

```
SQL> conn system/oracle
已连接。
SQL> create tablespace index_tbs
  2  datafile 'd:/index/index_tbs1.dbf'
  3  size 100M
  4  autoextend on;

表空间已创建。
```

此处创建表空间 INDEX_TBS 的目的就是存放索引，下面演示如何创建对表的多列创建索引，并且指定表空间。

例子 18-21　创建表多列索引

```
SQL> create index emp_ename_sal_idx
  2  on emp(ename,sal)
  3  tablespace index_tbs;

索引已创建。
```

我们依旧查看是否成功创建索引，使用数据字典 USER_INDEXES，如例子 18-22 所示。

例子 18-22　查看多列索引 EMP_ENAME_SAL_IDX 的信息

```
SQL> col index_name for a20
SQL> run
  1  select index_name,table_name,tablespace_name
  2  from user_indexes
  3* where index_name like 'EMP%'

INDEX_NAME           TABLE_NAME TABLESPACE
-------------------------------------- -------------------- -----------------
EMP_ENAME_IDX         EMP        USERS
EMP_ENAME_SAL_IDX     EMP        INDEX_TBS
```

从输出可以看出索引 MP_ENAME_SAL_IDX 是建立在表 EMP 上的，而且其存储表空间为 INDEX_TBS。但是如何查看一个索引是建立在表的哪几列上呢？答案是使用数据字典 USER_IND_COLUMNS。

在直观理解了如何创建单列索引和多列索引后，我们给出创建索引的语法格式。

```
CREATE [UNIQUE|BITMAP] INDEX [schema.] index_name
ON [schema.]table_name
(column_name[DESC]ASC][, column_name[DESC]ASC]]……)
[REVERSE]
[TABLESPACE tablespace_name]
[PCTFREE n]
[INITRANS n]
[MAXTRANS n]
```

```
[instorage state]
[LOGGING|NOLOGGING]
[NOSORT]
```

下面解释各个参数的含义：

- UNIQUE：说明该索引是唯一索引。
- BITMAP：创建位图索引。
- DESC|ASC：说明创建的索引为降序或升序排列。
- REVERSE：说明创建反向键索引。
- TABLESPACE：说明要创建的索引所存储的表空间。
- PCTFREE：索引块中预先保留的空间比列。
- INITRANS：每一个索引块中分配的事务数。
- MAXTRANS：每一个索引块中分配的最多事务数。
- INSTORAGE STATE：说明索引中区段 EXTENT 如何分配。
- LOGGING|NOLOGGING：说明“要记录|不记录”索引相关的操作，并保存在联机重做日志中。
- NOSORT：不需要在创建索引时再按键值进行排序。

18.9　查看索引

数据字典 USER_IND_COLUMNS 使得我们可以很方便的查找一个索引所对应的列的信息。在前面已经建立了两个索引，一个是单列索引 EMP_ENAME_IDX，一个是多列索引 EMP_ENAME_SAL_IDX。我们给出示例 18-23 查询索引相关的列的信息。

例子 18-23　查询与索引列相关的信息

```
SQL> col column_name for a40
SQL> select index name,table name,column name
  2  from user ind columns
  3* where index name like 'EMP%'

INDEX_NAME           TABLE_NAME   COLUMN_NAME
--------------------------------------------------------------------------
EMP ENAME IDX        EMP          ENAME
EMP_ENAME_SAL_IDX    EMP          SAL
EMP_ENAME_SAL_IDX    EMP          ENAME
```

输出结果说明了索引对应的表名，和索引是基于哪些列创建的，如输出所示索引 EMP_ENAME_SAL_IDX 是基于表 EMP 的两列 SAL 和 ENAME 创建的。

当然数据字典 USER_INDEXES 也可以很方便地查看索引信息，我们再给出一个例子。

例子 18-24　使用数据字典 USER_INDEXES 查看索引信息

```
SQL> col dropped for a10
SQL> run
  1  select index_name,table_name,table_owner,dropped,tablespace_name
```

```
  2  from user_indexes
  3* where index_name like 'EMP%'

INDEX_NAME          TABLE_NAME TABLE_OWNE DROPPED   TABLESPACE
--------------------------------------------------------------------------------
EMP_ENAME_IDX         EMP        SCOTT      NO        USERS
EMP_ENAME_SAL_IDX     EMP        SCOTT      NO        INDEX_TBS
```

在查看了我们创建的两个索引的信息，以索引 EMP_ENAME_SAL_IDX 为例，该索引对应的表为 EMP，该表所属的用户为 SCOTT，并且 DROPPED 为 NO，其存储表空间为 INDEX_TBS。这里 DROPPED 的含义是，该索引是否是被删除的一个标记。在 Oracle 11g 中，当删除一个对象时，先把该对象放入垃圾箱而不是立即从库中删除，如果删除一个对象该对象就标记 DROPPED 为 YES。

其实在前面创建的索引是 B 树索引，B 树索引是 Oracle 默认的索引类型。在了解了如何创建索引和查看索引后，我们想知道是如何创建的索引类型，或希望了解除了 B 树索引外还有哪些索引类型，以及这些索引的原理和优缺点，这就是下面几节要详细讨论的内容。

18.10 B树索引

B 树索引是 Oracle 默认的索引类型，研究 B 树索引也可以帮助理解位图索引和反向键索引，所以本节花较多篇幅讲解 B 树索引。

18.10.1 B 树索引的工作原理

叶子节点包含索引的实际值和该索引条目的行 ID 即 ROWID。B 树索引的结构如图 18-3 所示。

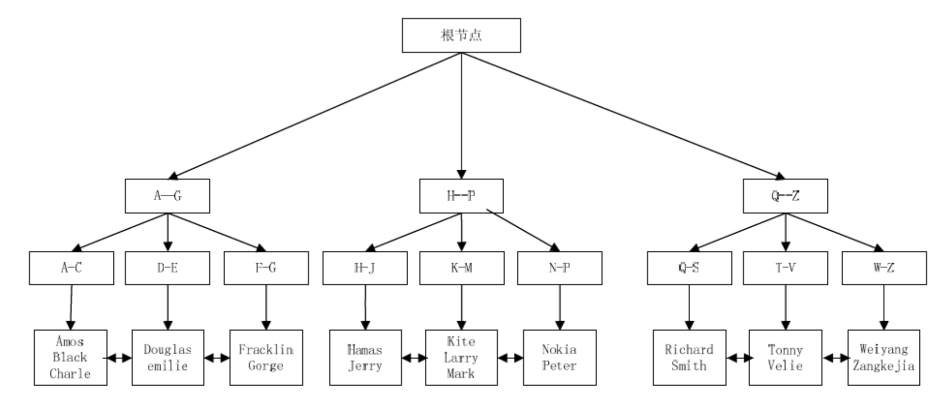


图 18-3 B 树索引结构图

如图 18-3 所示，B 树索引结构有 3 个基本组成部分：根节点、分支节点和叶子节点。其中根节点位于索引结构的最顶端，而叶子节点位于索引结构的最底端，中间为分子节点。

在叶子节点中存储了实际的索引列的值和该列所对应的记录的行 ID，即 ROWID，ROWID 是唯一的 Oracle 指针，指向该行的物理位置，使用 ROWID 是 Oracle 数据库中访问行最快的方法。

叶子节点其实是一个双向链表，每个叶子节点包含一个指向下一个和上一个叶子点的指针，这样在一定范围内便利索引以搜索需要的记录。

每个分支节点又包含其他分支节点，Oracle 设计的 B 树索引结构保证了 B 树索引从根到叶子都有相等的分支节点，保证了 B 树索引的平衡，这样就不会因为基表的数据插入后删除操作造成 B 树索引变得不平衡，从而影响索引的性能。并且如果一个叶子节点为空，则 Oracle 会释放该空间用于它处。

下面给出一个使用 B 树索引的搜索过程，如要查找 Larry，则在根节点转向中间的分支节点，然后继续搜索其下的分支节点，发现需要继续转到它的中间那个子分支节点即 K-M 节点，然后在叶子节点中找到所需要的列的值及其该列所对应的行 ID。从而找到更多的需要的数据。



Oracle 创建的普通索引如果没有说明类型就是 B 树索引。

18.10.2　B 树索引的注意事项

B 树索引在 Oracle 中是一个通用索引，创建索引的时候默认就是 B 树索引。可以是单列索引，也可以是组合索引(最多可以多达 32 个列)，对于 B 树索引，我们需要关注他的二元高度(blevel)。B 树索引保存了在索引列上有值的每个数据行的 ROWID。

Oracle 不会对索引列上包含 NULL 值的行进行索引，如果索引是一个组合索引，而其中列上包含 NULL 值，这一行会包含于索引列中。

18.11　位图索引

位图索引是 Oracle 11g Enterprise Edition 支持的索引机制。位图索引使用位图标识被索引的列值，它适用于没有大量更新任务的数据仓库，因为使用位图索引时，每个位图索引项与表中大量的行有关联，当表中有大量数据更新、删除和插入时，位图索引相应地需要做大量更改，而且索引所占用的磁盘空间也会明显增加，并且索引在更新时受影响的索引需要锁定，所以位图索引不适合于有大量更新操作的 OLTP 系统，虽然可以通过重建索引类位图索引，但是对于有大量更新操作的表最好不选择使用位图索引。

18.11.1　位图索引的使用讨论

位图发挥最大威力的场合是：当一个表中包含了多个位图索引，Oracle 就可以合并从每个位图索引得到的结果集，快速删除不必要数据。对于较低基数的位图索引来说，位图索引的尺寸远远小于 B 树索引，因此可以大大减少 I/O 的数量。

对于位图索引的列，列值的数量要求较少或者中等（索引列基数较小）。如列的基数是 4，Oracle 为每个唯一键创建一个位图，然后把与键值所关联的 ROWID 保存为位图。最多可以包括 30 列。

对于非常大的表来说，在多个低基数列上建立位图索引是一个很好的选择。对于位图索引来说，即使从表中读取很多行，也会使用位图索引。例如在一个 sex 列上建立索引，每次从表中读取

半数的数据行，但是还是会使用位图索引。

18.11.2　创建位图索引

下面我们以一个 SQL 查询为例子，解释位图索引的过程，该语句为：

```
SELECT EMPNO,ENAME,SAL
FROM EMP
WHERE JOB = 'SALESMAN';
```

上述查询语句的目的是在 EMP 表中查询工作岗位是 SALESMAN 的员工的员工号，姓名和薪水，此时假设已经在 EMP 表的 JOB 列建立了位图索引，其结构如图 18-4 所示。

Index on JOB

JOB =' CLERK'	1 0 0 0 0 0 0 0 0 0 1 1 0 1
JOB –' SALESMAN'	0 1 1 0 1 0 0 0 0 1 0 0 0 0
JOB =' PRESIDENT'	0 0 0 0 0 0 0 0 1 0 0 0 0 0
JOB =' MANAGER'	0 0 0 1 0 1 1 0 0 0 0 0 0 0
JOB =' ANALYST'	0 0 0 0 0 0 0 1 0 0 0 0 1 0

图 18-4　位图索引结构图

在该索引图中，共用 5 类 JOB，每类 JOB 对应 14 个比特位（对应 14 行记录），其中某行的在该列的值与 JOB 值对应则使用比特 1 表示，如 JOB = 'CLERK'，第一行在该列对应的值是 CLERK，就用比特 1 表示。否则用比特 0 表示，其他 JOB 类类似。

图 18-5 是位图索引操作的逻辑视图，通过该视图读者可以更清楚地了解 SQL 语句执行时，如何使用位图索引。

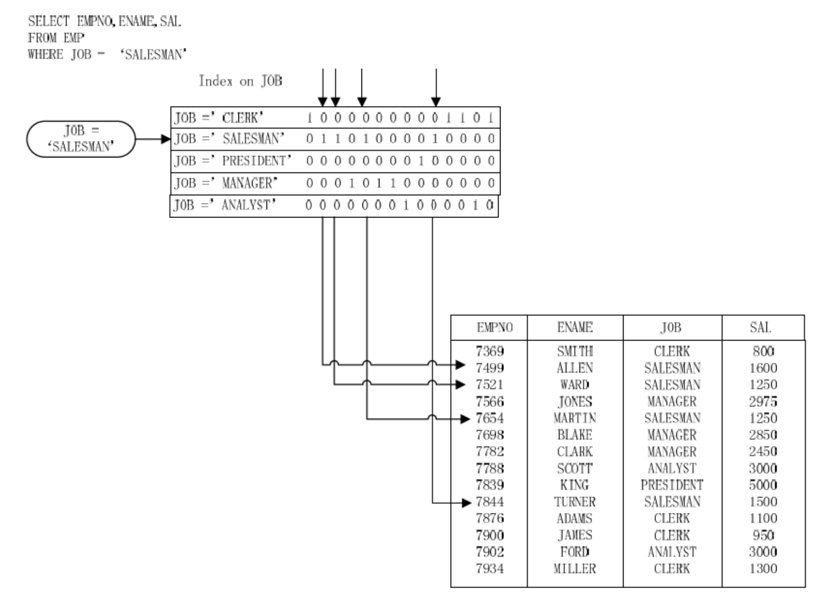


图 18-5　位图索引逻辑结构图

通过位图索引扫描 JOB='CLERK' 对应的位图记录，找到值为 1 的行记录，即找到需要查找

的数据。

下面给出一个例子来创建位图索引，如例子 18-25 所示。

例子 18-25 创建位图索引

```
SQL> create bitmap index emp job bitmap idx
  2  on emp(job);

索引已创建。
```

此时，我们成功创建了位图索引 EMP_ENAME_BITMAP_IDX。该索引基于表 EMP 的 ENAME 列创建。下面我们通过例子 18-26 查看该索引信息，主要是关注其类型标识。

例子 18-26 查看位图索引

```
SQL> col index name for a218
SQL> col index type for a118
SQL> col table_name for a10
SQL>select index_name,index_type,table_name,status
  2  from user indexes
  3* where index name like 'EMP%'

INDEX_NAME                INDEX_TYPE      TABLE_NAME STATUS
------------------------------------------------- ----------------
EMP_JOB_BITMAP_IDX         BITMAP          EMP         VALID
```

例子的输出说明创建的位图索引信息，其中 INDEX_TYPE 为 BITMAP，说明这是一个位图索引。

18.11.3 B 位图索引的插入问题

当使用位图索引的插入数据时，以下 3 点需要注意：

- 位图索引在批处理（单用户）操作中加载表（插入操作）方面通常比 B 树做得好。
- 当有多个会话同时向表中插入数据行时不应该使用位图索引。
- 当每条记录都增加一个新值时，B-树索引要比位图索引快 3 倍。

```
select ....
from participant
where age_code = 'B'
and income_level = 'DD'
and education_level = 'HS'
and marital_status = 'M';
```

上面查询语句例子中，4 个低基数列分别建立了位图索引。Oracle 会使用这 4 个位图索引对数据进行筛选，计算出需要读取的数据行和数据块，然后进行读取。在这个过程中会涉及位图的计算。

```
SQL> create bitmap index dept_idx2_bm on dept(dname);

索引已创建。

SQL> select index_name,index_type from user_indexes;

INDEX_NAME                      INDEX_TYPE
------------------------------ ---------------------------
PK_EMP                          NORMAL
IDX_SAL_EMP                     NORMAL
IDX_HIRE_EMP                    NORMAL
EMP_ID1                         NORMAL
PK_DEPT                         NORMAL
DEPT_IDX2_BM                    BITMAP

已选择6行。
```

在 B 树索引中，可以实现行级锁定，但是在位图索引中，因为对 ROWDI 进行压缩存放（一个 ROWID 范围+位图），因此每次锁定的都是整个的 ROWID 范围，因此对表中的位图索引列进行更新的时候，并发性很差，容易导致死锁。SELECT 语句不会受到这种锁定问题的影响。

位图索引有很多的限制：

- 基于规则的优化器不会考虑位图索引。
- 当执行 alter table 语句并修改包含位图索引列时，会使得位图索引失效。
- 位图索引不包含任何列数据，不能用于任何类型的完整性检查，例如主键、唯一键约束。
- 位图索引不能被声明为唯一索引。
- 位图索引的最大长度为 30。



不要在繁忙的 OLTP 系统中使用位图索引

18.12　Hash索引

使用 Hash 索引必须要使用 Hash Cluster。我们首先来看一下 Cluster 表的结构，如图 18-6 所示。

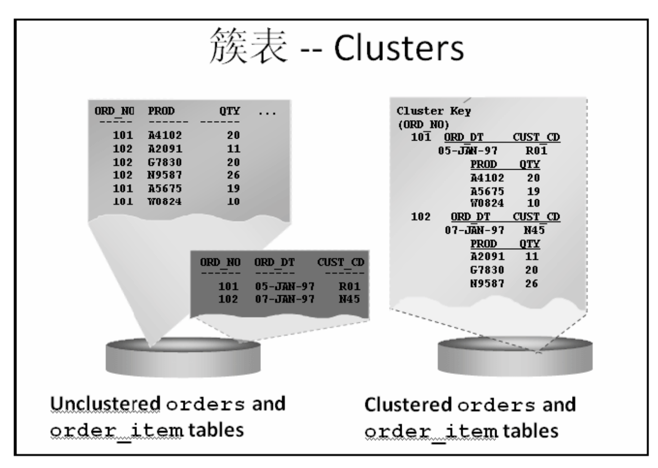


图 18-6　簇表结构

在存储数据时，所有与这个集群键相关的行都存储在一个数据块上。如果数据都存储在同一个数据块上，并且将 Hash 索引作为 where 子句的确切匹配条件，Oracle 就可以通过执行一个 Hash 函数和一个 I/O 来访问数据。如果通过一个二元高度是 3 的 B 树索引来访问数据则需要在检索数据时使用 4 个 I/O，如图 18-7 所示。

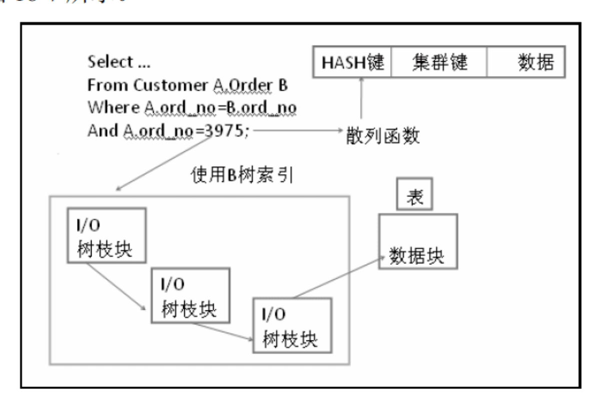


图 18-7

查询过程变换为等价查询，然后匹配 Hash 列和确切的值，最后基于 Hash 函数确定行的物理存储位置。

例子 18-27　创建 Cluster 表

```
create cluster credit_cluster(card_no varchar2(16),transdate date sort)
hashkeys 10000 hash is ora hash(card no) size 2186;

create table credit orders(card no varchar2(16),transdate date,amount number)
cluster credit_cluster(card_no,transdate);
```


这里的集群键列为 card_no、transdate，而 Hash 列为 card_no。

Hash 索引可能是访问数据库中数据的最快方法，但是有自身的缺点：

- 集群键上不同值的数目必须在创建 Hash 集群之前就需要确定，需要在创建 Hash 集群的时候指定这个值，低估了集群键的不同值的数字可能会导致集群的冲突（两个集群键有相同的 Hash 值）。
- 一旦这个值设置过低，需要重建 Cluster。
- Hash 集群还可能浪费空间，如果无法确定需要多少空间来维护某个集群键上的所有行，就可能造成空间浪费。
- 如果不能为集群的未来增长分配好附加的空间，Hash 集群可能就不是最好的选择。
- 如果应用程序经常在集群上进行全表扫描，Hash 集群可能不是最好的选择，由于需要为未来增长分配好集群的剩余空间，全表扫描可能非常地消耗资源，Hash 索引非常适合数据仓库（相对静态值）。

18.13　反向键索引

反向键索引是指在创建索引过程中对索引列创建的索引键值的字节反向，使用反向键索引的好处是将值连续插入到索引中时反向键能避免争用。

反向键索引适用于一种特殊的情形，如果一个索引值是按照序列值递增的，这样当连续插入大量数据时，所有的记录都将插入 B 树索引结构中的最右侧的叶子节点，并且会写入同一叶子节点中，这样难以避免产生争用问题而影响索引性能。正是为了避免这个问题引入了反向键索引。使用反向键索引使得每个键值被颠倒顺序，将序列性的键值分散开，使得键值平衡地保存在叶子节点中。如图 18-8 所示为键值颠倒的示意图。

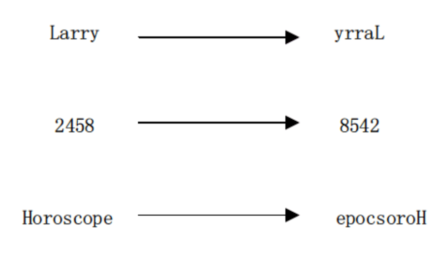


图 18-8　反向键索引的键值

创建反向键索引需要使用 REVERSE 关键字。

例子 18-28　创建反向键索引

```
SQL> create index emp_sal_reverse_idx
  2  on emp(sal) reverse;

索引已创建。
```

同样，我们通过数据字典 USER_INDEXES 查看刚刚创建的反向键索引信息，读者需要注意 INDEX_TYPE 列的值看 Oracle 是如何标识反向键索引类型的。

例子 18-29　通过数据字典视图查看反向键索引信息

```
SQL>  select index_name,index_type,table_name
  2   from user_indexes
  3  where index_name like 'EMP%';

INDEX_NAME               INDEX_TYPE               TABLE_NAME
---------------------------------------------------------------------------
EMP_ENAME_BITMAP_IDX     BITMAP                   EMP
EMP_SAL_REVERSE_IDX      NORMAL/REV               EMP
```

18.14　基于函数的索引

在用户查询数据时，如果查询语句的 WHERE 子句中有函数存在，Oracle 使用函数索引将加

快查询速度。基于函数的索引使用表的列的函数值作为键值建立索引结构，下面说明如何使用 UPPER 函数创建基于函数的索引。

例子 18-30　创建基于 UPPER 函数的函数索引

```
SQL> create index dept dname idx
  2  on dept(UPPER(dname));

索引已创建。
```

如上所示，我们创建了一个基于表 DEPT 中列 DNAME 的函数索引，创建该索引时首先将列 DNAME 中的值转换成大写，然后对大写的 DNAME 创建索引，放入索引表。这样当用户需要如下的查询：

```
SELECT UPPER(DNAME) FROM DEPT WHERE UPPER(DNAME) ='NEW YORK'
```

这时，Oracle 就不必对 WHERE 子句的条件做转化并逐行检索，对于选择的结果也不必使用 UPPER 函数再做转换的计算，显然此时使用基于函数的索引会极大地提高查询速度，如果该表很大的话，性能的提高是很明显的。

例子 18-31　通过数据字典 USER_INDEXES 查看基于函数的索引信息

```
SQL> col index_type for a30
SQL> select index name,index type,table name
  2  from user indexes
  3* where index name like 'DEPT%'

INDEX NAME          INDEX TYPE                    TABLE NAME
--------------------------------------------------------------------------
DEPT DNAME IDX       FUNCTION-BASED NORMAL          DEPT
```

上述输出中，同样读者需要注意 INDEX_TYPE 的值来区分索引类型，索引 DEPT_DNAME_IDX 的类型 INDEX_TYPE 为 FUNCTION_BASED NORMAL，说明它是基于函数的正常索引，该索引是在表 DEPT 上创建的。

例子 18-32　通过数据字典 USER_IDX_COLUMNS 查看基于函数的索引信息

```
SQL>  select index name,table name,column name
  2   from user_ind_columns;

INDEX NAME               TABLE NAME COLUMN NAME
---------------------------------------------------------------------------
PK DEPT                   DEPT        DEPTNO
PK_EMP                    EMP         EMPNO
EMP_ENAME_BITMAP_IDX      EMP         ENAME
EMP SAL REVERSE IDX       EMP         SAL
DEPT_DNAME_IDX            DEPT        SYS_NC00004$
```

从例子 18-32 的输出可以看出，索引 DEPT_DNAME_IDX 的 COLUMN_NAME 为系统赋予的一个值，因为这个列不是 Oracle 可以使用明确的列名标识的，它是某个列的函数，所以 Oracle 就使用自己的方法来标识 GOLUMN_NAME 列名。

基于函数的索引需要考虑的问题：

- 基于函数的索引只能针对一种函数，对于其余的函数不起作用。
- 控制索引的数量，因为对 DML 会产生影响。

18.15 监控索引的使用

虽然创建了索引，但是需要监控索引的使用情况，否则就无法判断我们创建的索引的有效性。对于无效的索引可以删除以释放磁盘空间并较少 DML 操作带来的修改索引的各种开销。

下面我们通过一个例子说明如何监控索引的使用，该方法在 Oracle 9i 以上中都适用，首先我们确定要监控索引 EMP_ENAME_BITMAP_IDX 索引是否被使用，此时我们需要执行如下的操作以启动监控行为。

例子 18-33 启动对索引 EMP_ENAME_BITMAP_IDX 的监控

```
SQL> alter index EMP_ENAME_BITMAP_IDX
  2  monitoring usage;

索引已更改。
```

接下来，我们要等待一个周期，在这个过程中等待用户对表 EMP 的各种操作，对于 OLTP（联机事务处理）系统这个周期可能很短，而对于数据仓库则需要更多的时间来监控。

下面我们模拟这个周期中的两个查询。

例子 18-34 模拟查询

```
SQL> select empno,ename,sal
  2  from emp
  3  where ename like 'S%';

     EMPNO ENAME             SAL
---------- ---------- ----------
      7369 SMITH             800
      7788 SCOTT            3000

SQL> select ename
  2  from emp;

ENAME
----------
ADAMS
ALLEN
BLAKE
………
WARD

已选择 14 行。
```

然后，我们终止对索引 EMP_ENAME_BITMAP_IDX 的监控，如例子 18-35 所示。

例子 18-35　终止监控 EMP_ENAME_BITMAP_IDX 索引

```
SQL> alter index EMP ENAME BITMAP IDX
  2  nomonitoring usage;

索引已更改。
```

现在，就可以使用数据字典视图 v$object_usage 查看 EMP_ENAME_BITMAP_IDX 索引的使用情况。

例子 18-36

```
SQL> select index_name, table_name, monitoring, used
  2  from v$object_usage;

INDEX NAME              TABLE NAME MON USE
---------------------------------------------------- -------------------
EMP_ENAME_BITMAP_IDX  EMP         NO  YES
索引已更改。
```

输出说明索引 EMP_ENAME_BITMAP_IDX 是基于表 EMP 创建的，当前没有监控该索引，因为 MON 为 NO，该索引已经被 Oracle 使用过了，因为 USED 为 YES。

在数据字典视图 v$object_usage 中，还提供了 START_MONITORING 和 END_MONITORING，来说明某个索引的监控周期，即起始时间和结束时间，如例子 18-37 所示。

例子 18-37　查看索引 EMP_ENAME_BITMAP_IDX 的监控周期

```
SQL> select index_name,start_monitoring,end_monitoring
  2  from v$object_usage
  3  where index name = 'EMP ENAME BITMAP IDX';

INDEX NAME              START MONITORING   END MONITORING
-------------------------------------------- --------------------------
EMP_ENAME_BITMAP_IDX  07/07/2012 20:42:16       07/07/2013 20:43:28
```



Oracle 也提供其他工具来监控索引使用的有效性，如使用 EXPLAIN PLAN 的输出和使用 SQL trace 等工具，有兴趣的读者可以参看相关书籍，这里不再详述。

18.16　重建索引

索引需要维护，不然如果建立了索引的表中有大量的删除和插入操作，会使得索引很大，因为在删除操作后，删除值所占用的索引空间不能被索引自动重新使用，而插入操作会不断使得索引变大，对于大表和 DML 操作很频繁的表，索引的维护是很重要的。Oralce 提供了一个 REBUILD 指令来重建索引，使得索引空间可以重用删除值所占有的空间，使得索引更加紧凑，如例子 18-38 所示。

例子 18-38　重建索引

```
SQL> alter index emp ename bitmap idx
  2  rebuild;
```

```
索引已更改。
```



使用索引重建不影响用户使用索引，但是有一些限制条件如不能使用 DDL 操作和 DML 操作。

在重建索引时也可以使用其他参数该表要重建的索引的表空间，该例子要求读者必须事先创建了表空间 INDEX_TBS1，如例子 18-39 所示。

例子 18-39　重建索引并迁移其表空间

```
SQL> alter index dept dname idx
  2  rebuild
  3  tablespace index_tbs1

索引已更改。
```

为了验证我们的重建索引 DEPT_DNAME_IDX 的结果，我们给出例子 18-40。

例子 18-40　使用数据字典 USER_INDEXES 验证重建索引有效性及表空间迁移变化

```
SQL> select index name,table name,tablespace name ,status
  2  from user indexes;

INDEX_NAME            TABLE_NAME TABLESPACE_NAME      STATUS
------------------------------------------------------------------------
PK EMP                   EMP       USERS                  VALID
EMP ENAME BITMAP IDX  EMP       USERS                  VALID
EMP SAL REVERSE IDX    EMP       USERS                  VALID
PK_DEPT                  DEPT      USERS                  VALID
DEPT_DNAME_IDX         EMP       INDEX_TBS1             VALID
```

通过查询输出可以看出索引 DEPT_DNAME_IDX 的状态 STATUS 为 VALID，说明该索引重建后是有效的，可以使用。而 TABLESPACE_NAME 为 INDEX_TBS1 说明已经成功迁移了索引 DEPT_DNAME_IDX 到表空间 INDEX_TBS1。

在重建索引时，还可以修改索引的其他参数，如 PCTFREE 或者 STORAGE 子句等，如例子 18-41 所示。

例子 18-41　重建索引 EMP_ENAME_BITMAP_IDX 并修改存储参数

```
SQL> alter index EMP ENAME BITMAP IDX
  2  rebuild
  3  pctfree 30
  4  storage (next 100k);

索引已更改。
```

也可以使用联机重建索引的方式，这样的方式重建索引用户可以执行 DML 操作，但是不能使用 DDL 操作，如例子 18-42 所示。

例子 18-42　联机重建索引

```
SQL> alter index dept dname idx
  2  rebuild online;

索引已更改。
```

18.17　维护索引

维护索引就是修改索引的各种参数，在维护索引前我们先需要知道当前索引的参数设置，如例子 18-43 所示。

例子 18-43　查询当前索引的参数设置

```
SQL> col index_name for a20
SQL> select index_name,pct_free,pct_increase,initial_extent,next_extent
  2* from user indexes

INDEX_NAME          PCT_FREE PCT_INCREASE INITIAL_EXTENT NEXT_EXTENT
--------------------------------------------------------------------------
PK EMP                   10                   65536
EMP SAL REVERSE IDX     10                   65536
EMP JOB BITMAP IDX       10                   65536
PK_DEPT                  10                   65536
```

从输出看出索引 EMP_JOB_BITMAP_IDX 的 PCT_FREE 为 10%，INITIAL_EXTENT 为 65 536 字节。下面我们通过例子 18-44 演示如何通过 REBUILD 修改参数。

例子 18-44　通过 REBUILD 修改索引参数

```
SQL> alter index emp job bitmap idx
  2  rebuild
  3  pctfree 30
  4  storage (next 100k);
```

下面再查看修改结果，使用数据字典 USER_INDEXES，如例子 18-45 所示。

例子 18-45　查询修改后索引 EMP_JOB_BITMAP_IDX 的参数

```
SQL> select index_name,pct_free,pct_increase,initial_extent,next_extent
  2  from user_indexes
  3  where index name = 'EMP JOB BITMAP IDX';

INDEX NAME         PCT FREE PCT INCREASE INITIAL EXTENT NEXT EXTENT
-------------------------------------------------------------------------
EMP_JOB_BITMAP_IDX  30                 65536

索引已更改。
```

此时，索引 EMP_JOB_BITMAP_IDX 的 PCT_FREE 参数确实修改为 30%，但是注意 NEXT_EXTENT 仍然是默认值，没有修改。因为索引 EMP_JOB_BITMAP_IDX 存储在表空间

USERS 中，而该表空间是本地管理的表空间索引无法修改 NEXT_EXTENT 参数。

接下来我们再演示如何为索引手工分配磁盘空间，如例子 18-46 所示。

例子 18-46　手工增加索引磁盘空间

```
SQL> alter index EMP_JOB_BITMAP_IDX
  2  allocate extent;

索引已更改。
```

Oracle 对于每个索引默认的 EXTENT 区段数为 1，此时为索引 EMP_JOB_BITMAP_IDX 增加了一个区段，所以该索引应该有两个区段。通过例子 18-47 查询该索引的参数信息。

例子 18-47　查询索引 EMP_JOB_BITMAP_IDX 的区段信息

```
SQL> select segment_name,segment_type,tablespace_name,extents
  2  from user_segments
  3  where segment_type = 'INDEX'
  4  and segment_name like 'EMP%';

SEGMENT_NAME       SEGMENT_TYPE    TABLESPACE_NAME      EXTENTS
-------------------------------------------------------------------------
EMP_JOB_BITMAP_IDX     INDEX            USERS                        2
EMP_SAL_REVERSE_IDX    INDEX            USERS                        1
```

从上述输出可以看出，索引 EMP_JOB_BITMAP_IDX 的区段 EXTENTS 数量为 2，说明我们已经成功为其添加区段。

另一种重要的维护索引的方式是合并索引碎片，其实合并碎片也是维护磁盘空间的方式。这里我们演示如何合并索引碎片，如例子 18-48 所示。

例子 18-48　合并索引碎片

```
SQL> alter index emp_job_bitmap_idx coalesce;

索引已更改。
```

通过合并索引碎片可以释放部分磁盘空间，是索引维护的一个重要方法。

18.18　删除索引

如果经过索引监控发现一个索引无效，或者处于效率考虑暂时不需要可以删除该索引，删除索引使用 DROP INDEX 指令，要求用户具备找个权限即可，如例子 18-49 所示。

例子 18-49　删除索引

```
SQL> drop index dept_dname_idx;

索引已删除。
```

在删除该索引后，为了确认删除结果，我们再使用下例通过数据字典 USER_INDEXES 查询删除结果。

例子 18-50 查看是否删除索引 DEPT_DNAME_IDX

```
SQL> select index name,index type,table name
  2  from user_indexes
  3  where index_name like 'DEPT%';

未选定行
```

显然我们要查找的索引不存在，说明已经成功删除了索引 DEPT_DNAME_IDX。

18.19 本章小结

本章重点是索引的原理以及各类索引的原理和使用。本章开始部分介绍 Oracle 实现数据访问的方法，索引是其中非常重要和常用的方法，然后介绍了索引的扫描类型，以及限制索引使用的条件。集群因子、二元高度、直方图都是索引使用中要注意的问题。在本章最后介绍了 Oracle 支持的索引监控以及索引维护。

第 19 章

系统和对象权限管理

权限管理是 Oracle 实现安全管理的一部分，通过授予不同用户的系统权限和对象权限实现用户对系统功能以及数据库对象的操作，本章分别讲解系统权限和对象权限，并通过实例说明如何授予和回收用户对象权限和系统权限。

19.1 权限的概念和分类

权限是执行特殊 SQL 语句或访问其他用户拥有的对象的权利，这些权利包括：连接数据库、维护表空间、改变系统状态、创建表、从用户表中选择数据行、执行存储过程等。

Oracle 将权限分为系统权限和用户权限。

- 系统权限：系统权限允许用户执行一个或一类特殊的数据库操作，如创建数据库、创建用户、创建与维护表空间以及管理会话等。
- 对象权限：对象权限是用户维护数据库对象的权利，如维护表、视图、序列号、存储过程、函数等。

19.2 系统权限

在 Oracle 数据库中有一类具有最高权限的用户，它可以实现对数据维护的任何工作，即 DBA 用户。DBA 用户可以为新用户或其他用户授权以执行某种操作，如赋予 SCOTT 用户访问所有对象的表的权利，赋予或回收执行系统功能的权利，直接将权限赋予用户或角色以及将权限赋予所有用户（PLUBLIC）。

在 Oracle 数据库中有 100 多种系统权限，在权限中的 ANY 关键字说明在任何模式中当前被授权的用户都具有这种权限，如 SELECT ANY TABLE 说明可以选择任何模式对象，GRANT 指令向用户或一组用户赋予某种权利，如 GRANT SELECT ANY TABLE TO SCOTT，就是向用户 SCOTT 赋予查看任何表的权利，而 REVOKE 指令说明要删除某个特权，如 REVOKE SELECT ANY TABLE FROM SCOTT，就是从 SCOTT 用户回收查看所有表的权利。

下面我们介绍常用的几类系统权限。

（1）与索引相关。

- CREATE ANY INDEX：创建任何模式中对象的索引。

- ALTER ANY INDEX：修改任何模式的索引。
- DROP ANY INDEX：删除任何模式的索引。



没有 CREATE INDEX 的权限。在 CREATE TABLE 权限中包含了 CREATE INDEX 的权利。

（2）与表相关。

- CREATE TABLE：在当前模式中创建表。
- CREATE ANY TABLE：在任何模式中创建表。
- ALTER ANY TABLE：修改任何模式中的表。
- DROP ANY TABLE：删除任何模式中的表。
- SELECT ANY TABLE：查看任何模式中的表数据。
- UPDATE ANY TABLE：修改任何模式中的表数据。
- DELETE ANY TABLE：删除任何模式中的表。



具有 CREATE TABLE 权限的用户自然拥有删除该表对象的权限，其他如 CREATE PROCEDURE、CREATE FUNCTION 类似。在创建表时必须为表分配表空间配额，或者对表空间具有无限制使用权利，如 UNLIMITED TABLESPACE。

（3）与会话相关。

- CREATE SESSION：建立数据库会话的权利。
- ALTER SESSION：修改数据库会话的权利。



一个新用户创建后，首先需要授予 CREATE SESSION 权限使它可以访问数据库。

（4）与表空间相关。

- CREATE TABLESPACE：创建表空间。
- ALTER TABLESPACE：修改表空间。
- DROP TABLESPACE：删除表空间。
- UNLIMITED TABLESPACE：允许使用所有表空间的权限。

19.3　授予用户系统权限

要授予用户系统权限需要使用 GRANT 指令的 SQL 语句，被授予了权限的用户在一定授权下可以继续将系统权限赋予其他用户。下面是赋予用户权限的语法格式。

```
    GRANT { system privilege | role }
          [, { system privilege | role } ] ......
       TO   { user | role | PUBLIC }
```

```
          [, { user | role | PUBLIC } ] ……
       [ WITH ADMIN OPTION ]
```

如上所示整个授权的框架是：GRANT 系统权限 TO 用户 [WITH ADMIN OPTION]。如果有多个系统权限使用逗号隔开，如果有多个用户也用逗号隔开。在上述语法格式中符号“|”表示“或”的关系。

下面通过例子演示如何向用户授权。首先，我们创建一个用户 JNAE。

例子 19-1 创建新用户 JANE

```
SQL> create user jane
  2  identified by abc119#;

用户已创建。
```



例子 19-1 中的密码有点奇怪，因为我们使用了 verify_function 密码复杂性验证函数，且更改了用户的默认密码概要文件，所以这里的密码要符合设置的规则。

对于一个新创建的数据库用户，它不具有任何权限，如果此时使用该用户连接数据库则无法成功，如例子 19-2 所示。

例子 19-2 使用新用户 JANE 连接数据库

```
SQL> connect jane/abc119#@orcl
ERROR:
ORA-01045: user JANE lacks CREATE SESSION privilege; logon denied


警告：您不再连接到 ORACLE。
```

该例子说明用户 JNAE 没有 CREATE SESSION 的系统权限，所以登录被拒绝了。下面我们将 CREATE SESSION、CREATE TABLE、SELECT ANY TABLE 的权限赋予用户，此时必须以 SYSTEM 用户登录。

例子 19-3 赋予用户权限

```
SQL> grant create session,create table,select any table to jane;

授权成功。
```

现在用户 JANE 具有了建立数据库会话、创建表以及查看任何表的系统权限，下面通过数据字典 DBA_SYS_PRIVS 查看被授权的用户的权限信息。

例子 19-4 查看用户 JANE 的拥有的系统权限

```
SQL> conn system/oracle@orcl
已连接。
SQL> col grantee for a10
SQL> col privilege for a25
SQL> select *
  2  from dba_sys_privs
```

```
  3* where grantee = 'JANE'

GRANTEE    PRIVILEGE                  ADM
-------------------- ---------------------------------------- ----------
JANE       SELECT ANY TABLE            NO
JANE       CREATE TABLE               NO
JANE       CREATE SESSION              NO
```

从输出可以看出用户 JANE 具有了 3 个系统权限，即 SELECT ANY TABLE、CREATE TABLE 和 CREATE SESSION。而 ADM 列的值都为 NO，说明用户 JANE 拥有的权限不能再赋予其他用户。为了验证整个问题，我们再创建一个新用户 LARRY。

例子 19-5　创建用户 LARRY

```
SQL> create user larry
  2  identified by abc119#;

用户已创建。
```

新用户创建成功，我们使用数据字典 DBA_SYS_PRIVS 查看该用户的系统权利是不是“一穷二白”。

例子 19-6　查看用户 LARRY 的系统权限

```
SQL> select *
  2  from dba sys privs
  3  where grantee = 'LARRY';

未选定行
```

显然，用户 LARRY 目前还不具有任何系统权限，下面我们尝试使用 JANE 用户登录并将 CREATE SESSION 和 SELECT ANY TABLE 的权限赋予用户 LARRY。

例子 19-7　用户 JANE 赋予用户 LARRY 系统权限

```
SQL> connect jane/abc119#@orcl
已连接。
SQL> grant create session,select any table to larry;
grant create session,select any table to larry
*
第 1 行出现错误:
ORA-01031: 权限不足
```

发生错误，说明权限不够，用户 JANE 无法向用户 LARRY 授权。读者还记得在例子 19-4 中用户 JANE 的 ADM 列的值都为 NO，所以无法继续授权给其他用户。下面我们修改用户 JANE 的权限使得它具有继续授权给其他用户的权利。先回收用户权限。

例子 19-8　回收用户 JANE 的所有权限

```
SQL> connect system/oracle@orcl
已连接。
SQL> revoke create session,select any table,create table from jane;
```

```
撤销成功。
```

下面重新授予用户 JANE 这些权利，并带有 WITH ADMIN OPTION 选项，如例子 19-9 所示。

例子 19-9　赋予用户 JANE 系统权限并允许继续授权

```
SQL> grant create session,select any table,create table to jane
  2  with admin option;

授权成功。
```

为了验证 WITH ADMIN OPTION 的参数设置效果，我们继续使用数据字典 DBA_SYS_PRIVS。

例子 19-10　查看用户 JANE 的系统权限信息

```
SQL> select *
  2  from dba_sys_privs
  3  where grantee ='JANE';

GRANTEE    PRIVILEGE                   ADM
------------------- ---------------------------------------- -----------
JANE       SELECT ANY TABLE             YES
JANE       CREATE TABLE                YES
JANE       CREATE SESSION              YES
```

此时用户 JANE 系统权限的 ADM 列的值都为 YES，说明这些权限可以继续赋予其他用户。下面将用户 JANE 的 CREATE SESSION 和 SELECT ANY TABLE 赋予用户 LARRY。

例子 19-11　用户 JANE 赋予用户 LARRY 系统权限

```
SQL> connect jane/abc119#@orcl
已连接。
SQL> grant create session,select any table to larry;

授权成功。
```

授权成功，我们通过数据字典 DBA_SYS_PRIVS 查看用户 LARRY 具有的系统权限。

例子 19-12　查看用户 LARRY 的系统权限

```
SQL> connect system/oracle@orcl
已连接。
SQL> select *
  2  from dba sys privs
  3  where grantee ='LARRY';

GRANTEE    PRIVILEGE                   ADM
-------------------- --------------------------------------- ----------
LARRY      CREATE SESSION              NO
LARRY      SELECT ANY TABLE             NO
```

从输出可以清楚地看出用户 LARRY 具有了 CREATE SESSION 和 SELECT ANY TABLE 的权利，但是用户 LARRY 不能将这些权利再赋予其他用户，因为在向用户 LARRY 授权时，没有使用 WITH ADMIN OPTION 选项。

下面，我们使用 LARRY 用户登录数据库并查询 SCOTT 用户的表信息。

例子 19-13　使用用户 LARRY 登录数据库

```
SQL> connect larry/abc119#@orcl
已连接。
SQL> select *
  2  from scott.dept;

    DEPTNO  DNAME          LOC
---------------------------------------- --------------
        40  OPERATION       BOSTON
        10  ACCOUNTING     NEW YORK
        20  RESEARCH       DALLAS
        30  SALES           CHICAGO
```

因为用户 LARRY 被赋予了 CREATE SESSION 的权利，所以可以成功连接数据库，而 SELECT ANY TABLE 的权限使得它可以查看任何用户的表信息。

因为我们创建的用户都是用于连接数据库，并查看一些表信息，如果业务允许我们可以事先将一些权限赋予当前所有的用户，如 CREATE SESSION、SELECT ANY TABLE 等，如例子 19-14 所示。

例子 19-14　将部分系统权限赋予所有用户

```
SQL> conn system/oracle@orcl
已连接。
SQL> grant create session,select any table to public;

授权成功。
```

19.4　SYSDBA和SYSOPER系统特权

Oracle 提供了两个特殊的系统权限，即 SYSDB 权限和 SYSOPER 权限，在做系统维护时建议使用这两种系统权限登录数据库。用户通过 SYSDBA 连接到数据库时，它具有对数据库的一切特权。下面是两种系统特权的典型数据库操作。

与 SYSDBA 系统特权相关的操作：

- SYSOPER PRIVILEGES WITH ADMIN OPTION：具有 SYSOPER 所具有的操作，并且可以将这些特权赋予其他用户。
- CREATE DATABASE：创建数据库。
- ALTER DATABASE BEGIN/END BACKUP：将数据库置于备份状态。
- RESTRICTED SESSION：设置会话限制。
- RECOVER DATABASE UNTIL：介质恢复数据库到 UNTIL 指定的状态。

例子 19-15　使用 SYSDBA 特权登录数据库

```
SQL> connect system/oracle@orcl as sysdba;
已连接。
```

与 SYSOPER 系统特权相关的操作：

- STARTUP：启动数据库。
- SHUTDOWN：关闭数据库。
- ALTER DATABASE OPEN | MOUNT：将数据库切换到打开|挂起状态。
- ALTER DATABASE BACKUP CONTROLFILE TO：备份控制文件。
- RECOVER DATABASE：介质恢复数据库。
- ALTER DATABASE ARCHIVELOG：将数据库设置为归档模式。

19.5 回收用户系统权限

如果需要限制某个用户的权限可以回收权限，使用 REVOKE 指令，回收用户权限的语法格式如下所示。

```
REVOKE {system privilege | role }
     [, {system_privilege | role ] ……
FROM { user | role | PUBLIC }
     [,{user | role | PUBLIC} ] ……
```

下面我们再查询用户 JANE 和 LARRY 的用户系统权限，如例子 19-16 所示。

例子 19-16　查询用户 JANE 和 LARRY 的系统权限

```
SQL> col grantee for a10
SQL> col privilege for a25
SQL> col admin option for a15
SQL> select *
  2  from dba_sys_privs
  3  where grantee IN ('JANE','LARRY')
  4* order by grantee

GRANTEE    PRIVILEGE                 ADMIN_OPTION
----------------------------------------------------------------------
JANE       CREATE SESSION            YES
JANE       CREATE TABLE              YES
JANE       SELECT ANY TABLE          YES
LARRY      CREATE SESSION            NO
LARRY      SELECT ANY TABLE          NO
```

从以上输出可以看出，用户 JANE 和 LARRY 都具有系统权限，下面演示如何回收用户 LARRY 的所有系统权限，如例子 19-17 所示。

例子 19-17　回收用户 LARRY 的系统权限

```
SQL> connect system/oracle@orcl
已连接。
SQL> revoke create session,select any table
  2  from larry;

撤销成功。
```

撤销成功说明回收了用户 LARRY 的系统权限，下面我们验证回收结果。

例子 19-18　查询用户 LARRY 的系统权限

```
SQL> select *
  2  from dba_sys_privs
  3  where grantee ='LARRY';

未选定行
```

输出结果说明，数据字典 DBA_SYS_PRIVS 中没有记录用户 LARRY 的系统权限信息，说明例子 19-17 成功回收用户 LARRY 的系统权限。

对于授权时使用了 WITH ADMIN OPTION 选项的用户的权限回收工作需要做一些说明，所以再次为用户 LARRY 赋予 CREATE SESSION 和 SELECT ANY TABLE 的权利，并带 WITH ADMIN OPTION 选项，并创建一个新用户 SOPHIE。

例子 19-19　为用户 LARRY 赋予系统权限并带 WITH ADMIN OPTION 选项

```
SQL> grant create session,select any table
  2  to larry with admin option;

授权成功。
```

创建新用户 SOPHIE。

```
SQL> create user sophie
  2  identified by abc119#;

用户已创建。
```

此时使用 LARRY 用户登录数据库，然后将 CREATE SESSION 和 SELECT ANY TABLE 的权限赋予用户 SOPHIE 如下所示。

例子 19-20　用户 LARRY 向用户 SOPHIE 授以系统权限

```
SQL> conn larry/abc119#@orcl
已连接。
SQL> grant create session,select any table
  2  to sophie;

授权成功。
```

此时，用户 SOPHIE 具有 CREATE SESSION 和 SELECT ANY TABLE 的系统权限，为了验证结果，我们再次使用数据字典 DBA_SYS_PRIVS。

例子 19-21　查询用户 SOPHIE 的系统权限信息

```
SQL> conn system/oracle@orcl
已连接。
SQL> col grantee for a10
SQL> col privilege for a25
SQL> col admin_option for a15
SQL> select *
```

```
  2  from dba sys privs
  3* where grantee ='SOPHIE'

GRANTEE    PRIVILEGE                  ADMIN OPTION
-------------------------------------------------------------------------
SOPHIE     CREATE SESSION              NO
SOPHIE     SELECT ANY TABLE            NO
```

此时，用户 JANE 向用户 LARRY 授以系统权限，用户 LARRY 具有继续向其他用户授予系统权限的能力，此时用户 LARRY 又向新用户 SOPHIE 授予系统权限，那么如果用户 JANE 回收了用户 LARRY 的 SELECT ANY TABLE 的权利，是否影响用户 SOPHIE 的 SELECT ANY TABLE 的系统权限呢？我们通过例子说明。

例子 19-22　用户 JANE 回收用户 LARRY 的 SELECT ANY TABLE 权限

```
SQL> connect jane/abc119#@orcl
已连接。
SQL> revoke select any table
  2  from larry;

撤销成功。
```

输出显示成功回收用户 LARRY 的 SELECT ANY TABLE 系统权限，接着我们验证用户 SOPHIE 是否还具有 SELECT ANY TABLE 权限。我们使用用户 SOPHIE 登录数据库。然后查询表信息。

例子 19-23　使用用户 SOPHIE 登录数据库并查询表信息

```
SQL> connect sophie/abc119#@orcl
已连接。
SQL> select *
  2  from scott.dept;

    DEPTNO DNAME          LOC
-------------- --------------------------- -------------
        40 OPERATION      BOSTON
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
```

显然，虽然用户 LARRY 的 SELECT ANY TABLE 系统权限被回收了，但是用户 SOPHIE 仍然具有 SELECT ANY TABLE 权利。

例子 19-23 中关于 WITH ADMIN OPTION 选项相关的权限回收示例为了说明 REVOKE 回收系统权限不具备级联特性。

读者是否还记得，我们赋予任何用户 PUBLIC 以 CREATE SESSION 和 SELECT ANY TABLE 的权利，下面我们回收用户的这些系统权限。

例子 19-24　回收授予所有用户的系统权限

```
SQL> connect system/oracle@orcl
已连接。
```

```
SQL> revoke create session,select any table
  2  from public;

撤销成功。
```

19.6　授予对象权限

和系统权限相对应的是对象权限，对象包括表、视图（物化视图）、序列号和存储过程等。在这些数据库对象上实现某种特殊的行为的权限称为对象权限，如对于一个表可以更改表的结构、删除表或更新表中的数据行等。Oracle 的对象权限包括：ALTER、DELETE、EXECUTE、INDEX、INSERT、REFERENCES、SELECT 和 UPDATE。而这些对象权限适用于不同的数据库对象。

表 19-1 是对数据库对象的权限与对应的数据库对象关系的一个列表。

表 19-1　对象权限列表

对象权限	表（table）	视图（view）	序列号(sequence)	过程(procedure)
ALTER	y		y	y
DELETE	y	y		
EXECUTE				y
INDEX	y	y		
INSERT	y	y		
REFERENCES	y			
SELECT	y	y	y	
UPDATE	y	y		

在上述对象的权限列表中，SEQUENCE 序列号只有两种对象权限，即 ALTER 和 EXECUTE。而对于对象权限 ALTER、UPDATE、REFEREMCES 和 INSERT 可以实现更小粒度的权限控制，如可以对表对象的某列加以限制等。

在给出一个具体的对象权限授予的例子前，给出对象授权的语法格式：

```
GRANT {object_privilege [ ( column_list ) ]
      [, object privilege [ ( column list) ] ] ……
      | ALL [ PRIVILEGE ]
ON    [schema. ] object
TO    { user | role | PUBLIC } [, { user | role |PUBLIC } ]……
      [ WITH GRANT OPTION]
```

- GRANT：授权关键字。
- Object_privilege：对象权限。
- Column_list：对象权限操作的列的列表。
- All：将当前用户的某个数据库对象的所有权限赋予新用户。
- On object：说明具体的数据库对象，如表或存储过程。
- With grant option：新用户可以继续授权。

接下来，我们给出一个将表对象权限 UPDATE 赋予新用户 LARRY 的例子，并且 LARRY 用户可以继续将该对象权限赋予其他用户。

例子 19-25　把 SCOTT 用户的 EMP 表象权限 UPDATE 赋予新用户 LARRY

```
SQL> connect scott/tiger@bjyzz
已连接。
SQL> grant update on emp
  2  to larry with grant option;

授权成功。
```

例子 19-25 中，我们使用 SCOTT 用户登录数据库，并且将 SCOTT 用户 EMP 表的 UPDATE 权限赋予用户 LARRY。我们通过数据字典 USER_TAB_PRIVS_MADE 来查看对象权限的授权信息。

例子 19-26　查看 SCOTT 用户中表对象的授权信息

```
SQL> col table name for a10
SQL> col grantor for a15
SQL> col privilege for a10
SQL> select *
  2* from user tab privs made

GRANTEE    TABLE_NAME GRANTOR         PRIVILEGE  GRA
--------------------------------------------- ----------------- ---------
LARRY       EMP        SCOTT            UPDATE    YES
```

从输出可以清楚地看出 GRANTOR 是 SCOTT 用户，而 GRANTEE 是 LARRY 用户，SCOTT 将拥有的表 EMP 的 UPDATE 对象权限赋予了 LARRY。

下面我们将 SCOTT 用户的某个表的某些列的对象权限赋予用户 SOPHIE，如例子 19-27 所示。

例子 19-27　把对表 DEPT 的列的 UPDATE 对象权限赋予用户 SOPHIE

```
SQL> grant update(dname,loc) on dept to sophie;

授权成功。
```

Oracle 提供了一个数据字典 USER_COL_PRIVS_MADE 记录用户的列对象权限的赋予情况，如例子 19-28 所示。

例子 19-28　使用数据字典 USER_COL_PRIVS_MADE 查看相关列的权限赋予信息

```
SQL> col column_name for a10
SQL> select *
  2  from user col privs made;

GRANTEE    TABLE_NAME COLUMN_NAM GRANTOR    PRIVILEGE  GRA
-------------------------- ------------------- ------------------ -------
SOPHIE     DEPT       DNAME      SCOTT          UPDATE     NO
SOPHIE     DEPT       LOC         SCOTT          UPDATE     NO
```

从上述输出可以看出，当前用户 SCOTT 的表中列的权限赋予信息，其中表为 DEPT 而与其相

关的列为 DNAME 和 LOC，将两列的 UPDATE 权限赋予用户 SOPHIE。

在本节，我们使用 SCOTT 用户将表 EMP 的 UPDATE 权限赋予了用户 LARRY，且用户 LARRY 可以将该权限继续赋予其他用户，而同时又将 DEPT 表的列 DNAME 和 LOC 赋予用户 SOPHIE。

19.7　回收对象权限

出于安全的考虑，如果一个用户不需要某种对象权限可以使用 REVOKE 指令回收用户的对象权限。如下是回收对象权限的语法格式：

```
REVOKE { object privilege
         [, object_privilege ] ……
         | ALL [ PRIVILEGE ] }
ON   [schema.] object
FROM   {user | role | PUBLIC}
         [, { user | role | PUBLIC } ] ……
         [ CASCADE CONSTRAINTS ]
```

下面回收用户 LARRY 和 SOPHIE 的对象权限，如例子 19-29 所示。

例子 19-29　回收用户 LARRY 对 EMP 表的 UPDATE 对象权限

```
SQL> conn scott/tiger@bjyzz
已连接。
SQL> revoke update on emp
  2  from larry ;

撤销成功。
```



例子 19-29 中我们使用 SCOTT 用户登录数据库，然后我们使用数据字典 USER_TAB_PRIVS_MADE 来验证是否成功回收权限。

例子 19-30　验证是否成功回收赋予用户 LARRY 的对象权限

```
SQL> select *
  2  from user_tab_privs_made
  3  where grantee ='LARRY';

未选定行
```

显然，数据字典 USER_TAB_PRIVS_MADE 中没有记录用户 SCOTT 的表对象权限授权信息。说明成功回收赋予用户 LARRY 的对象权限。

例子 19-31　回收用户 SOPHIE 对 DEPT 表的列操作的权限

```
SQL> revoke all on dept
  2  from sophie;

撤销成功。
```

在回收对象权限时，只能从整个表而不能按列回收。所以虽然我们按照列赋予用户 SOPHIE

的对象权限，但是不能按列权限回收。以下是一个错误示例。

例子 19-32　一个错误示例

```
SQL> revoke update(dname,loc) on dept
  2  from sophie;
revoke update(dname,loc) on dept
             *
ERROR 位于第 1 行:
ORA-01750: UPDATE/REFERENCES 仅可以从整个表而不能按列 REVOKE
```

在例子 19-31 中，我们成功回收了用户 SOPHIE 对表 DEPT 的列 DNAME 和 LOC 的 UPDATE 权限。下面通过例子验证是否成功回收权限。

例子 19-33　验证是否成功回收拥护 SOPHIE 的对象权限

```
SQL> select *
  2  from user col privs made;

未选定行
```

从输出结果可以看出，在数据字典 USER_COL_PRIVS_MADE 中没有记录用户 SOPHIE 的对象操作权限。



对象权限的回收是级联的，如用户 SCOTT 授予用户 LARRY 对象权限 A 且具有继续授权能力，用户 LARRY 继续将对象权限 A 赋予用户 SOPHIE 且具有继续授权能力，用户 SOPHIE 可以继续授权，如果此时用户 SCOTT 回收用户 LARRY 的对象权限 A 则用户 SOPHIE 不在具有继续向其他用户授予对象权限 A 的能力。

19.8　本章小结

本章主要讲了数据库系统权限和对象权限管理，系统权限是数据库系统相关的权限，如创建数据库、创建表空间等，它是 Oracle 数据库中具有最高权限的用户，它可以实现对数据维护的任何工作。读者需要掌握授予和回收系统权限，对象权限指对数据库对象如表、视图、序列号和存储过程等操作的权利，这些权利包括 ALTER、UPDATE、DELETE 和 INSERT 等，在维护数据库对象时需要读者很好地掌握这些对象权限对应的数据库对象类型，并掌握如何授予和回收对象权限。

第 20 章

视　图

本节讲述普通视图和物化视图。普通视图是一个虚表，不占用存储空间，在数据字典中只有视图的定义，视图可以通过 DML 语言操作，但是有一定限制，因为操作视图最终还是操纵创建视图的底层表。物化视图是 Oracle 10g 中提出的概念，物化视图是一种占用存储空间的特殊视图，物化视图完成数据复制、数据同步、数据汇总以及数据发布，物化视图在生产数据库中很有实用价值，尤其在分布式计算环境和移动计算环境中，我们将通过介绍物化视图的概念、如何创建物化视图以及在使用物化视图中遇到的一些问题详细介绍物化视图。

20.1　什么是视图

首先，视图是一种虚表，它不存储数据，在 Oracle 的数据字典中只是记录了视图的定义，视图通过 select 语句定义。

在多表查询中为了简化查询过程，创建基于多表的一个查询视图，用户每次通过查询视图类查询所需要的数据。用户可以查询视图甚至使用 UPDATE、DELETE 或者 INSERT 语句操纵视图，但是此时操纵的还是基表中的数据，切记：视图只有定义没有物理存储，在操作视图时实际上是通过执行 SQL 语句操纵定义视图的实际的物理表。

20.2　创建视图

我们通过例子来体验和说明如何创建视图。我们使用 SCOTT 用户的表来创建视图，但是必须授予 SCOTT 用户创建视图的权限。授权方法如例子 20-1 所示。

例子 20-1　授予 SCOTT 用户创建视图的权限

```
SQL> conn system/oracle as sysdba
已连接。
SQL> grant create view to scott;
```

在 SCOTT 用户模式下，有一个表对象 EMP，即员工表，该表记录了员工号、员工名字、工作性质、雇佣时间、薪水以及部门号等。为了方便每个部门查询自己部门内部员工信息，我们为每个部门创建一个视图，这样不同的部门只要使用视图就可以完成查询，而不用再使用多表连接和

WHERE 条件语句来限制查询的部门（虽然实际的操作还是一样“复杂”，但至少对使用者简单），并且可以通过和表 DEPT 的联合查询给出该部门的名字。我们创建属于 ACCOUNTING 部门的员工视图，如例子 20-2 所示。

例子 20-2　创建属于 ACCOUNTING 部门的员工视图

```
    SQL>  create view accounting_view as
      2   select e.ename "employee name",e.job "job",e.hiredate "hiredate",e.sal
"salary",d.dname "dep name"
      3   from dept d,emp e
      4   where e.deptno =d.deptno
      5  and d.deptno <20;

    视图已创建。
```

从上述创建视图的例子可以看出，使用 CREATE VIEW viewname AS 语句创建视图，AS 后是 SQL 查询语句，一旦视图创建成功，则在数据字典中会记录该视图的定义，如例子 20-3 所示，我们查询数据字典中记录的视图定义。

例子 20-3　查询数据字典中记录的视图定义

```
    SQL> select view_name
      2  from user_views;

    VIEW_NAME
    --------------------
    ACCOUNTING_VIEW
```

我们再查询该视图的定义语句，在视图 USER_VIEWS 中使用 TEXT 属性列记录该视图的定义，如例子 20-4 所示。

例子 20-4　查询视图 ACCOUNTING_VIEW 的定义

```
    SQL> select text
      2  from user_views
      3  where view_name = 'ACCOUNTING_VIEW';

    TEXT
    ------------------------------------------------------------------------
    select   e.ename   "employee name",e.job   "job",e.hiredate   "hiredate",e.sal
"salary",
```

创建视图后，再需要查询关于 ACCOUNTING 部门的员工信息时，就可以通过视图 ACCOUNTING_VIEW 来实现，并且该视图的列对基表进行了重命名。我们查询 ACCOUNTING 部门的所有员工信息，如例子 20-5 所示。

例子 20-5　查询 ACCOUNTING 部门的所有员工信息

```
    SQL> select *
      2  from accounting_view;

    employee_n job         hiredate          salary dep_name
```

```
--------------------------------------------------------------------------
CLARK   MANAGER   09-6 月 -81      2450 ACCOUNTING
KING     PRESIDENT  17-11 月-81      5000 ACCOUNTING
MILLER   CLERK      23-1 月 -82      1300 ACCOUNTING
```

我们也可以通过以下方式创建视图，比如此时创建部门 SALES 的员工表，如例子 20-6 所示。

例子 20-6　创建部门 SALES 的员工视图

```
SQL>  create or replace view sales_view
  2  ("employee name","job","hiredate","salary","dep name")
  3  as
  4  select e.ename,e.job,e.hiredate,e.sal,d.dname
  5  from dept d,emp e
  6  where e.deptno = d.deptno
  7  and d.deptno = 30;

视图已创建。
```

我们同样查询是否成功创建 SALES_VIEW 视图，如例子 20-7 所示。

例子 20-7　查询是否成功创建 SALES_VIEW 视图

```
SQL> select view name,text
  2  from user_views
  3* where view_name like 'SAL%'

VIEW NAME
------------------------------
TEXT
------------------------------------------------
SALES VIEW
select e.ename,e.job,e.hiredate,e.sal,d.dname
from dept d,emp e
where e.deptno =
```

从例子 20-7 的输出可以看出，成功创建视图 SALES_VIEW。

在例子 20-6 中，我们使用了 replace 参数，并且将别名放在了视图名字的后面，这样做是允许的。虽然与例子 20-2 创建视图的方式不同，但都是正确的创建方式。当需要查询部门 SALES 的员工信息时，就可以直接使用 SALES_VIEW 视图，而不必使用复杂的查询语句，如例子 20-8 所示。

例子 20-8　使用视图的查询

```
SQL> select *
  2  from sales_view;

employee n    job         hiredate     salary, dep name
--------------------------------------------- ---------- --------------
ALLEN      SALESMAN  20-2 月 -201     1600 SALES
WARD       SALESMAN  22-2 月 -201     1250 SALES
MARTIN     SALESMAN  220-9 月 -201     1250 SALES
BLAKE      MANAGER   01-5 月 -201     22050 SALES
TURNER     SALESMAN  020-9 月 -201     1500 SALES
JAMES      CLERK       03-12 月-201     950 SALES
```

```
已选择 6 行。
```

下面总结创建视图的语法格式，如下所示。

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view name
[别名[,别名]……]
AS
```

查询子句

```
[WITH CHECK OPTION [CONSTRAINT 约束名]]
[WITH READ ONLY]
```

下面依次介绍每个选项：

- CREATE OR REPLACE：创建视图，如果所创建的视图名存在用新创建的视图覆盖原视图。
- FORCE/NOFORCE：FORCE 说明创建视图时，不论基表是否存在都创建该视图；NOFORCE 则相反，只有所引用的基表都存在时才创建该视图。
- 别名：就是所创建的视图的列名，数量与视图所产生的列的数量相等。
- AS：该关键字说明下面是查询子句，用户定义视图。
- 查询子句：是任意正确的完整的查询语句。
- WITH CHECK OPTION：当更新某一数据行时，必须满足 WHERE 子句的条件。
- WITH READ ONLY：设置该视图为“只读”状态，说明无法在该视图上进行任何 DML 操作。

20.3 使用视图的WITH子句

在创建视图的语法格式中，我们给出了完整的语法格式，注意有两个 WITH 子句的使用，使得视图更加安全。因为视图更多是给非专业人员使用的，所以要考虑到使用者可能的操作。

试想你创建了一个视图，而视图的使用者可以随意更改视图，修改某一列的值等，显然这样的操作会直接反映在创建这个视图的基表上，使得基表数据被改变，显然这是不合理的。而使用 WITH READ ONLY 子句就可以很好地解决找个问题，而 WITH CHECK OPTION 则可以增加约束条件，使得用户对数据的更新受到某些限制。下面依次讲解如何使用 WITH READ ONLY 子句和 WITH CHECK OPTION 子句。

- WITH READ ONLY 子句

类似于例子 20-6 我们再创建部门 RESEARCH 的员工信息，如例子 20-9 所示。

例子 20-9 创建部门 RESEARCH 的员工视图

```
SQL> create or replace view research view
  2  ("employee name","job","hiredate","salary","dep name")
  3  as
  4  select e.ename,e.job,e.hiredate,e.sal  ,d.dname
```

```
  5  from dept d,emp e
  6  where e.deptno = d.deptno
  7  and d.deptno = 20
  20  with read only;

视图已创建。
```

在该例子中，我们使用了 WITH READ ONLY 子句，也就是不允许对该视图使用 DML 操作。否则提示如例子 20-10 所示的错误。

例子 20-10　对该视图 research_view 使用 DML 操作的错误提示

```
SQL> update research_view
  2  set "salary" = 1000
  3* where  "job" = 'CLERK'
set "salary," = 1000
    *
第 2 行出现错误:
ORA-01733: 此处不允许虚拟列
```

再使用 DELETE 语句测试是否可以删除数据，如例子 20-11 所示。

例子 20-11　测试是否可以删除数据

```
SQL> delete from research view
  2  where "job" = 'CLERK';
delete from research view
           *
第 1 行出现错误:
ORA-01752: 不能从没有一个键值保存表的视图中删除
```

例子 20-10 和例子 20-11 说明通过 WITH READ ONLY 子句创建的视图不能使用 DML 语言。

- WITH CHECK OPTION 子句

使用该子句当通过视图更新数据或删除数据时不能违背 WHERE 子句限制的条件。如我们先创建一个视图，该视图只涉及一个表，如例子 20-12 所示。

例子 20-12　创建一个基于单表的视图

```
SQL> create view emp_view as
  2  select *
  3  from emp
  4  where JOB IN ('SALESMAN','MANAGER');

视图已创建。
```

例子 20-12 的 WHERE 子句限制只选择表 EMP 中 JOB 为 SALESMAN 和 MANAGER 的所有员工数据，即只有 JOB 为 SALESMAN 和 MANAGER 的数据才可以插入，否则不允许插入。

下面，我们试图插入一行员工数据，且 JOB 为 Marketing，如例子 20-13 所示。

例子 20-13　尝试向视图 emp_view 中插入一行员工数据

```
SQL> insert into emp_view(empno,ename,job,mgr,hiredate,
```

```
  2  sal,comm,deptno)
  3  values (7565,'TOM','Marketing',79920,SYSDATE,2000,22,40);
insert into emp_view(empno,ename,job,mgr,hiredate,
           *
第 1 行出现错误:
ORA-01402: 视图 WITH CHECK OPTIDN where 子句违规
```

此时，不允许插入该行数据，因为插入的数据违反了 WHERE 子句中 JOB IN ('SALESMAN','MANAGER')的条件。

但是如果插入如下一行数据是允许的，如例子 20-14 所示。

例子 20-14　向视图 emp_view 中插入一行员工数据

```
SQL> insert into emp view(empno,ename,job,mgr,hiredate,
  2  sal,comm,deptno)
  3* values (7565,'TOM','MANAGER',72039,SYSDATE,2000,22,20)

已创建 1 行。
```

显然例子 20-14 中的插入操作没有违反 WHERE 子句的约束，因为 JOB 为 MANAGER 为了确认插入结果使用例子 20-15 验证结果。

例子 20-15　确认例子 20-14 的插入结果

```
SQL> select *
  2  from emp view
  3  ename = 'TOM';
EMPNO ENAME   JOB        MGR  HIREDATE        SAL   COMM  DEPTNO
-------------------------------------------------------------------------
7565     TOM    MANAGER  72039  21-7 月 -10        2000    22    20
```

我们知道，操纵视图最终是通过视图的定义来操作实际的表，所以例子 20-15 的插入操作实际上会在表 EMP 中插入该行数据，我们用例子 20-16 来验证。

例子 20-16　查询表 EMP 中是否插入了一行数据

```
SQL> select *
  2  from emp
  3  ename = 'TOM';
EMPNO ENAME   JOB        MGR  HIREDATE        SAL   COMM  DEPTNO
--------------------------------------------- ---------- --------------
7565     TOM    MANAGER  72039  21-7 月 -10        2000    22    20
```

20.4　视图的修改

如果处于需要想修改视图的定义，此时 Oracle 提供的唯一方法就是重新定义该视图，用新定义的视图覆盖原来的视图。使用 CREATE OR REPLACE VIEW 子句。

在例子 20-9 中，我们创建了视图 RESEARCH_VIEW，该视图中包含了 RESEARCH 部门的所有员工信息，但是此时需要增加一个员工属性员工号 empno。首先，我们通过例子 20-17 查看视图

RESEARCH_VIEW 是否存在。

例子 20-17 查看视图 RESEARCH_VIEW 是否存在

```
SQL> select view name
  2  from user_views;

VIEW NAME
--------------------------
SALES_VIEW
RESEARCH_VIEW
ACCOUNTING_VIEW
```

为了确认我们查看该视图的列属性，如例子 20-18 所示。

例子 20-18 确认要查看的视图 RESEARCH_VIEW 的结构

```
SQL> desc research_view;
 名称                                    是否为空? 类型
 ----------------------------------------- --------- --------------
 employee_name                                       VARCHAR2(10)
 job                                                 VARCHAR2(9)
 hiredate                                            DATE
 salary,                                             NUMBER(7,2)
 dep_name                                            VARCHAR2(14)
```

视图 RESEARCH_VIEW 有 5 个列属性，但是没有员工号 EMPNO，现在，我们修改视图 RESEARCH_VIEW，增加员工号属性并且将所有的列名改为中文，如例子 20-19 所示。

例子 20-19 修改视图 RESEARCH_VIEW

```
SQL> create or replace view research_view
  2  ("员工号","员工姓名","岗位","雇佣时间","薪水","部门")
  3  as
  4  select e.empno,e.ename,e.job,e.hiredate,e.sal,d.dname
  5  from dept d , emp e
  6  where e.deptno = d.deptno
  7* and d.deptno = 20

视图已创建。
```

我们查询该视图的列名信息，看是否是当前我们创建的新视图的列名，如例子 20-20 所示。

例子 20-20 确认是否创建新视图

```
SQL> desc research view;
 名称                                    是否为空? 类型
 ------------------------------------------------------- ------------------
 员工号                                  NOT NULL NUMBER(4)
 员工姓名                                         VARCHAR2(10)
 岗位                                             VARCHAR2(9)
 雇佣时间                                         DATE
 薪水                                             NUMBER(7,2)
 部门                                             VARCHAR2(14)
```

显然，旧的视图已经被修改，下面，我们查询该视图的一些数据，看语句的输出效果，中文的列名对于用户来讲或许更友好☺，如例子 20-21 所示。

例子 20-21　查询视图 RESEARCH_VIEW 的信息

```
SQL> select *
  2  from research view
  3  where rownum<3;

      员工号 员工姓名    岗位          雇佣时间             薪水  部门
--------------- ---------------------------------- -------- -----------
         7565 TOM        MANAGER   21-7 月 -10        2000  RESEARCH
         7369 SMITH      CLERK      17-12 月-10          2000  RESEARCH
```

20.5　Oracle的视图管理

视图只是一个虚表，它不存储数据，对视图的操作最终是通过视图的定义操纵它所涉及的表，在成功创建视图后，在 Oracle 数据库的数据字典中记录该视图的信息。Oracle 使用数据字典管理视图，该数据字典是 USER_VIEWS。

20.5.1　通过数据字典查询视图

因为在 20.2 节已经使用了数据字典 USER_VIEWS，所以这里只给出一个例子，来查询当前我们已经创建的视图信息，如例子 20-22 所示。

例子 20-22　查询当前我们已经创建的视图信息

```
SQL> select view name
  2  from user_views;

VIEW NAME
------------------------------
SALES_VIEW
RESEARCH_VIEW
ACCOUNTING VIEW
EMP_VIEW
```

可以看出，我们已经创建了 4 个视图，其中前 3 个视图是基于表 EMP 和表 DEPT 创建的每个部门的员工信息视图，最后一个视图 EMP_VIEW 是使用单表 EMP 创建的视图。

20.5.2　Oracle 视图查询的内部过程

Oracle 在使用视图查询数据时，会执行一系列的分析过程，最终才执行 SQL 查询语句。整个过程如下：

- 读取数据字典，获得该视图的定义，查找到该视图所引用的表。
- 从数据字典中查询当前用户对于该视图所引用的表的权限。

- 执行定义该视图的 SQL 语句，实现执行视图查询。

为了更形象地理解视图查询的内部过程，给出如图 20-1 所示的过程图。

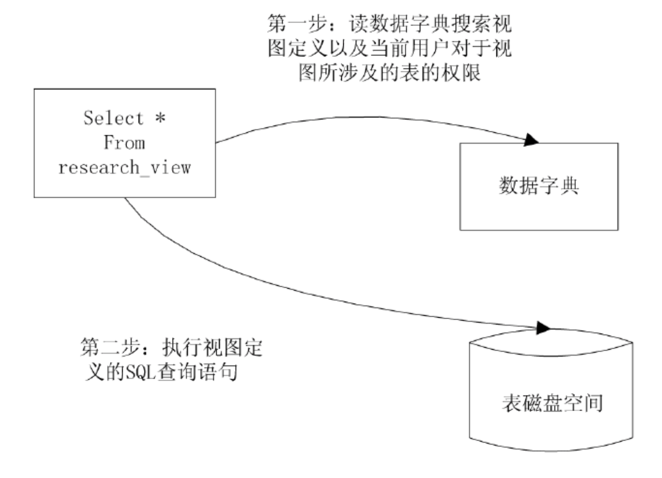


图 20-1　视图查询内部过程图



数据字典和表都保存在磁盘上，所以上述过程实际上要实现两次磁盘 I/O，这对系统的效率会有一定影响。

20.6　视图DML操作的限制

视图的 DML 操作是有限制的，毕竟视图的操作最终转化成对它引用的表的物理操作，所以依据视图的类型不同，和是否使用函数等条件对于视图的 DML 操作有一定的限制，下面我们分简单视图和复杂视图分别介绍它们在 DML 操作的限制。

20.6.1　简单视图

简单视图从一个表读取数据，不包括函数和分组数据。简单视图可以进行 DML 操作。即可以对简单视图进行 DELETE、UPDATE 和 INSERT 操作，简单视图的 DML 操作直接转化成对定义它的表的 DML 操作。

20.6.2　复杂视图

复杂视图从多个表提取数据，包括函数和分组数据，复杂视图不一定能进行 DML 操作。Oracle 对于在复杂视图上进行 DML 操作加了很多限制条件，即：

- 如果复杂视图包含了分组函数、GROUP BY 子句或者 DISTINCT 关键字，就不能使用 DELETE、UPDATE 或 INSERT 的 DML 操作。在复杂视图上进行 DML 操作最终也要转化成对视图所引用表的 DML 操作，由于复杂视图中包含函数或分组函数所以就不能在复

杂视图上使用 DML 操作。

- 如果复杂视图中的列包含表达式，或者有伪列 ROWNUM，则不能使用复杂视图进行 UPDATE 或 INSERT 等 DML 操作。

20.7　视图的优点

从以上几节的学习和示例中，或许读者已经体会到使用视图的好处，这里我们总结一下视图的优点。

- 减少数据操纵的复杂性：在执行基于多表的查询中，用户需要输入一长串的列名，使用表的别名，输入复杂的条件语句等，显然如果每次输入这样复杂的操作对于非专业人员是件“痛苦”的事，而使用视图，则可以简化这些语句输入，并且用一个易于记忆的名字命名视图。
- 增强安全性： 可以将视图设置为 READ ONLY，这样使用者就无法修改或更新数据，并且相同的数据可以显示在不同的视图中，如果一个表数据很重要但是用户需要访问该表中的某列值，则可以使用视图查询该重要表中的该列值，而不用查询这个表。
- 重命名列：很多表的列名是开发数据库的专业人员使用的，而对于非专业人员理解起来不直观，通过创建视图，重命名列达到使列名含义更清晰直观的效果。
- 实现数据定制：我们在本章创建的视图是基于不同的部门创建的员工表，这样一个部门内部的员工通过授权就可以只看到自己部门内部的员工信息，实现了定制的本单位员工数据。
- 保护数据的完整性：通过视图的 WITH CHECK OPTION 子句实现数据行的完整性约束和数据有效性检查。

20.8　删除视图

如果不需要一个视图，可以删除该视图，删除视图的指令是 DROP VIEW。删除视图 EMP_VIEW，如例子 20-23 所示。

例子 20-23　删除视图 EMP_VIEW

```
SQL> drop view emp view;

视图已删除。
```

一旦删除了视图，要使用数据字典 USER_VIEWS 验证是否删除，如例子 20-24 所示。

例子 20-24　使用数据字典 USER_VIEWS 验证是否删除

```
SQL> select view name
  2  from user_views;

VIEW NAME
----------------------------
```

```
SALES VIEW
RESEARCH_VIEW
ACCOUNTING_VIEW
```

显然该例子中没有了视图 EMP_VIEW 的定义，说明删除成功。

20.9　物化视图

本节我们讲物化视图，它是和普通视图相对应的概念，简单讲物化视图就是具有物理存储的特殊视图，占据物理空间，就像表对象一样。物化视图是基于表、物化视图等创建的。它需要和源表进行同步，不断地刷新物化视图中的数据，本节会讲到所有这些问题，并且使读者掌握创建物化视图的条件，以及如何创建物化视图。

20.9.1　什么是物化视图

在以上几节，我们重点讲解了视图的概念、使用和管理等，这些视图我们称为普通视图。在 Oracle 使用普通视图时，它会重新执行创建视图的所有 SQL 语句，如果这样的视图有多张表的 JION 或 ORDER BY 子句，而且表相当大，则会相当耗时。使用普通视图的查询效率很低。为了解决这个问题，Oracle 提出了物化视图的概念，物化视图是具有物理存储的特殊视图，它占用存储空间，可以进行分区和创建索引等操作。

物化视图是基于表、视图或者其他物化视图创建的。当创建一个物化视图时，Oracle 会自动创建一个内部表来存放物化视图的数据。

20.9.2　查询重写的概念

重写查询顾名思义是对 SQL 查询语句进行重写。当用户使用 SQL 语句对基表进行查询时，如果已经创建了基于这些基表的物化视图，Oracle 将自动计算和使用物化视图来完成查询，毕竟物化视图在某些情况下可以节约查询时间，减少系统 I/O。我们把 Oracle 的这种查询优化技术称为查询重写。

Oracle 提供基于成本的优化程序（CBO）将自动计算使用物化视图还是使用基于基表的查询，这些成本包括 CPU 开销、内存使用和系统 I/O 等。如果一个查询涉及多个表的连接操作，则使用物化视图可以避免连接操作使用的 CPU 和 I/O 开销，减少查询时间。

当然，Oracle 提供了灵活的方式使得用户自己选择是否使用查询重写功能，参数 QUERY_REWRITE_ENABLED 决定是否使用重写查询，该参数为布尔值（Boolean），默认参数值为 false，表示不执行查询重写。在创建物化视图时需要使用 ENABLE QUERY REWRITE 来启动查询重写功能。下面我们通过 SHOW 指令查看参数 QUERY_REWRITE_ENABLED 的值，如例子 20-25 所示。

例子 20-25　通过 SHOW 指令查看参数 QUERY_REWRITE_ENABLED 的值

```
SQL> show parameter query rewrite enabled

NAME                                TYPE       VALUE
------------------------------------------------------------ -----------
```

```
query_rewrite_enabled                    string      TRUE
```

参数 QUERY_REWRITE_ENABLED 的 VALUE 为 TRUE，说明当前运行的数据库允许查询重写功能。

提高查询性能是物化视图一大优点，Oracle 优化器就是通过代价计算来选择物化视图，通过查询重写来完成用户查询。优化器自动地计算判断一个物化视图是否能满足用户的查询要求，以及是否可以提高查询性能，如果满足要求且可以提高查询性能，优化器就重写用户提交的查询以使用物化视图，查询重写对用户是不可见的。

20.9.3 物化视图的同步

物化视图是基于基表创建的，所以当基表变化时，需要同步数据以更新物化视图中的数据，这样保持物化视图中的数据和基表中的数据的一致性。Oracle 提供了两种物化视图的刷新方式。即 ON COMMIT 方式和 ON DEMAND 方式。

使用 ON COMMIT 方式，当一个基表的变化提交时，则物化视图自动更新，完成与基表的同步。而使用 ON DEMAND 方式时，需要手动同步物化视图和基表数据，此时必须执行 DBMS_MVIEW.REFRESH 过程来同步物化视图。

在选择一种刷新方式后，就可以选择一种刷新类型完成数据同步，刷新类型指刷新数据时如何实现基表与物化视图的同步，从而将基表的变化反映在物化视图中， Oracle 提供了 4 种刷新类型。

- COMPLETE 类型：该类型将重新执行创建物化视图的 SQL 查询语句，无论基表中修改的数据量的多少，都需要完成一次对物化视图的重新计算。
- FAST 类型：该类型使用每个基表的物化视图日志只同步变化了的数据。显然这种方式将节约查询时间成本。
- FORCE 类型：该类型显示使用 FAST 类型更新数据，如果失败，则再使用 COMPLETE 类型刷新整个物化视图。
- NEVER 类型：从不更新，显然这个类型只对那些基表数据不变的物化视图有效。

如果在创建物化视图时，不指定一种刷新类型，则默认使用 FORCE 刷新类型。

图 20-2 给出使用 FAST 类型刷新数据的物化示意图。

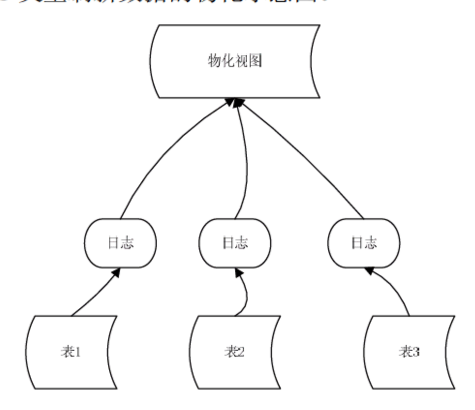


图 20-2 使用 FAST 类型同步物化视图

如图 20-2 所示，一个物化视图涉及 3 个表，表 1、表 2 和表 3，每个表有一个物化视图日志，把自己的变化记录在日志中，而物化视图则通过这些日志文件获得基表的变化，显然这种方式减少了重新执行物化视图的更新时间。



在启动查询重写机制后，Oracle 的查询重写也有可能无法实现，因为无法满足查询重写的某些条件，此时虽然创建了物化视图，但是数据库并不使用它，显然这样就失去了创建物化视图的作用，一旦这种情况发生，可以使用 DBMS_MVIEW 程序包中的过程进行分析。读者可以参考 Oracle 11g 文档查看该程序包中的过程和使用方法。

20.9.4　创建物化视图

用户可以根据自己的需要来创建物化视图，当然要求用户必须理解自己的表结构和数据需求。创建物化视图的目的是减少诸如普通视图中带有 JOIN 和 ORDER BY 子句带来的查询耗时问题，以减少系统资源的压力。

（1）创建物化视图的前提条件

首先介绍创建物化视图的前提条件，创建物化视图的用户必须具有创建物化视图的权限，QUERY REWRITE 的权限，以及对创建物化视图所涉及的表的访问权限和创建表的权限。下面，通过 SCOTT 用户来演示上述权限的赋予情况，如例子 20-26 所示。

例子 20-26　创建物化视图的用户授予权限

```
SQL> conn system/oracle as sysdba
已连接。
SQL> grant create materialized view to scott;

授权成功。

SQL> grant query rewrite to scott;

授权成功。
SQL> grant create any table to scott;

授权成功。

SQL> grant select any table to scott;

授权成功。
```

此时，我们使用 DBA 用户为 SCOTT 用户赋予了创建物化视图的所有权限。

（2）创建物化视图日志

物化视图日志是用户选择了刷新类型为 FAST 时要使用的，以同步基表的变化，所以，如果在创建物化视图时选择 FAST 刷新类型则需要创建该日志，我们计划对 SCOTT 用户的表 DEPT 和表 EMP 创建物化视图，所以对这两个基表创建物化视图日志，如例子 20-27 所示。

例子 20-27　针对基表创建物化视图日志

```
SQL> create materialized view log on dept;

物化化视图日志已创建。

SQL> create materialized view log on emp;

物化化视图日志已创建。
```

在创建完基表的物化视图日志后，就可以使用创建带有 REFRESH FAST 的参数创建物化视图了。

（3）创建物化视图的语句

创建物化视图的语句很简单，我们通过 CREATE MATERIALIEZDE VIEW 来创建物化视图，这里需要注意各个参数的含义。我们通过例子 20-28 来说明如何创建该视图以及各参数的含义。

例子 20-28　如何创建物化视图

```
SQL> create materialized view mtrlview test
  2  build immediate
  3  refresh fast on commit
  4  enable query rewrite
  5  as
  6  select d.dname,d.loc,e.ename,e.job,e.mgr,e.hiredate,e.sal
  7  from dept d,emp e
  20  where d.deptno = e.deptno;

物化化视图已创建。
```

下面详细介绍例子 20-28 中的各参数和子句的作用。

- BUILDE IMMEDIATE：该参数的含义是立即创建物化视图，BUILDE 的含义在一些编译环境中（如 Jbuilder）是编译的意思，此时含义类似，希望读者通过理解关键字记住参数的含义。与立即创建对应的自然是延迟创建的参数，即 BUILD DEFFERED ，该参数说明在物化视图定义后不会立即执行，而是延迟执行，在使用该视图时在创建。
- REFRESH FAST ON COMMIT：FEFRESH FAST 说明刷新数据的类型选择 FAST 类型，即快速刷新，该刷新类型要求使用物化视图日志实现与基表数据的同步。ON COMMIT 说明在基表数据提交后立即更新物化视图。
- ENABLE QUERY REWRITE：概述参数的含义是启动查询重写功能，我们已经知道 Oracle 对于查询重写功能的默认选项是不启用的，所以在创建物化视图时明确说明启用查询重写功能。
- AS 子句：该子句定义物化视图的内容，物化视图中存储该语句查询的结果存储在内部表中。
- 查询体：AS 之后的语句定义了物化视图的查询内容，该 SQL 语句的查询结果输出到物化视图中，保存在由 Oracle 自动创建的表中。

（4）删除物化视图

删除物化视图和删除普通视图相似，不过需要添加一个 MATERIALIZED 关键字，如例子 20-29 所示。

例子 20-29　删除物化视图

```
SQL> drop materialized view mtrlview test;

物化化视图已删除。
```

20.9.5　物化视图的使用环境

从以上对物化视图的定义、分析，以及创建和管理物化视图的实例中，我们已经体会到物化视图的基本作用，可以这样总结：物化视图是一种可以用于汇总、计算、复制以及发布数据的方案。与以上行为对应，物化视图适用于数据仓库、决策支持、分布式计算，以及移动计算等环境。

- 数据仓库：在数据仓库中往往需要存储对于基表的汇总或平均数据等，物化视图用户进行类似的计算来存储聚合后的数据。因为在数据仓库中的物化视图通常存储汇总数据，所以数据仓库中的物化视图也称为概要。
- 分布式环境：在分布式环境中可以通过物化视图实现不同节点间的数据同步，使得同样的数据分布在不同的物理空间，更好地响应用户的查询，减少中心数据库服务器的负担。物化视图的复制功能使得用户可以把数据复制到远程节点，而数据同步能力又可以同步各个节点间的数据，从而实现了数据的本地访问。为了形象说明物化视图在分布式环境中的应用，给出图 20-3。

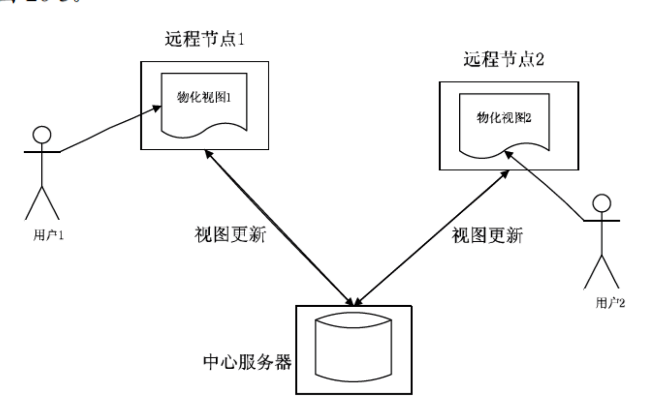


图 20-3　物化视图在分布式环境中的应用

在图 20-3 中无论用户 1 还是用户 2，他们查询中心服务器的数据时，都是通过查询本地的物化视图实现的，物化视图通过数据同步机制跟新数据，对于用户而言这个过程是透明的，用户认为是通过远程的中心服务器实现了该查询。

- 移动计算环境：该环境中也是利用物化视图的优化查询功能来节约查询时间，同时移动客户端使用物化视图下载一个数据子集，也可以定期地从中央服务器获得新数据，在客户端修改后发到中央服务器。

显然在生产数据库中物化视图是非常重要的应用，尤其在分布式环境和移动计算环境中，物化视图的分布式存储和同步功能很好地满足客户对于本地读取数据的需求，更好地响应客户对于数据请求的时间要求。

20.10　本章小结

视图极大地方便了用户对于数据的操作，不但增加了对表访问的安全性，而且减少了很多复杂的查询过程，本章我们重点讲了普通视图和物化视图，对于普通视图对于学过数据库原理之类课程的读者来说很容易理解，在本章前半部分重点讲了视图的概念如何创建和修改视图。Oracle 的视图管理部分使读者可以把握通过数据字典查看视图的方式，以及视图执行的内部过程。对于在普通视图上的 DML 操作 Oracle 做了限制，对于简单视图可以使用 DML 操作而对于复杂视图则必须遵守一定的原则或条件。

在本章的后半部分我们引入了物化视图，需要读者理解物化视图的本质，物化视图具有存储空间，在一定条件下可以实现 DML 操作，使用同步机制实现与创建物化视图的表的数据同步，物化视图在数据仓库、分布式计算和移动计算中都具有很重要的作用。

第 21 章

◀ 序列号和同义词 ▶

序列号和同义词是 Oracle 中两个很有用的对象，序列号生成器会自动管理序列号，对于某些订单系统很有用处。使用同义词对象可以创建很多对象，如表、索引、函数、过程等的同义词，使用同义词使得操作的对象更加易于理解，同时使用同义词可以方便用户访问属于其他用户的数据库对象，或出于安全目的，因为同义词名字可以随意命名，没有限制，这样就隐藏了创建同义词的原始对象的信息。

21.1　什么是序列号

Oracle 使用序列生成器自动产生用户可以在事务中使用的唯一序列号，该序列号是一个整数类型数据，序列生成器主要完成在多用户环境下产生唯一的数字序列，但是不会造成额外的磁盘 I/O 或事务锁。简单的说序列号是 Oracle 数据库的一个对象，该对象产生唯一序列号。

在不使用序列号时，如果多个用户同时向 EMP 表中插入一条员工记录，用户必须等待以得到下一个可用的员工号，而一旦使用序列号则用户无需相互等待就可以得到下一个可用的员工号。序列生成器会自动为每个用户创建正确的员工编号。

可见使用序列生成器，避免了多用户相互等待而造成的事务串行执行，序列号的使用提高了系统的事务处理能力，减少多用户并行操作的等待时间。

Oracle 的序列号有如下特点：

- *序列号是独立于表的对象，由 Oracle 自动维护。*
- *序列号可以对多个用户共享使用，即同一个序列对象供多个表使用且相互独立。*
- *在 SQL 语句中使用序列号就可以使用它产生的序列号。*

下面我们给出实例说明如何创建和使用序列号以及序列号的维护工作。

21.2　创建和使用序列号

本节我们讲解如何创建一个序列号，并给出使用序列号的实例分析序列号的特点。如例子 21-1 所示。

例子 21-1　创建序列号

```
SQL> create sequence emp seq
  2  start with 1000
  3  increment by 1
  21  nomaxvalue
  5  nocycle;

序列已创建。
```

提示“序列已创建”，此时我们使用数据字典 USER_SEQUENCES 来查看刚刚创建的序列号的信息，如例子 21-2 所示。

例子 21-2　使用数据字典 USER_SEQUENCES 查看序列号信息

```
SQL> col cycle flag for a10
SQL> select sequence name,min value,increment by,cycle flag
  2  from user_sequences
  3* where sequence_name like 'EMP%'

SEQUENCE NAME                 MIN VALUE  INCREMENT BY CYCLE FLAG
--------------------------------------------------- --------------------
EMP_SEQ                               1           1         N
```

例子 21-2 的结果说明我们刚刚创建的序列号 EMP_SEQ 已经记录在系统当中，接着就可以使用该序列号为我们服务了。

这里给出创建序列号的语句格式，使得读者可以全面了解创建序列号的各种参数以及含义。创建序列号的语句格式如下：

```
CREATE SEQUENCE sequence name
[START WITH n]
[INCREMENT BY n]
[{MAXVALUE n|NOMAXVALUE}]
[{MINVALUE n|NOMINVALUE}]
[{CACHE n|NOCACHE}]
[{CYCLE n|NOCYCLE}]
```

下面依次解释各参数的含义：

- START WITH n：序列号初始值，默认值为 1。
- INCREMENT BY n：序列号的步进幅度，默认步进幅度为 1。
- MAXVALUE n：定义序列号的最大值。
- NOMAXVALUE：不设置序列号的最大值，但是我们知道计算机对于数据的表达是有限的，实际上这个值当序列号是升序时为 10^{27}，序列号为降序时为-1。
- MINVALUE n：定义序列号的最小值。
- NOMINVALUE：不定义序列号的最小值，但是和参数 NOMAXVALUE 对应该值对于升序的序列号最小值为 1，对于降序的序列号的最小值为-10^{26}。
- CACHE n：Oracle 服务器会预分配 n 个序列号并保保存在内存中。
- NOCACHE：Oracle 服务器不会预分配序列号并保保存在内存中。

- CYCLE n：定义序列号在达到最大值或最小值后，将继续产生序列号。
- NOCYCLE：定义序列号在达到最大值或最小值后，不再产生序列号。

在使用序列号前，先创建一个表 EMPLOYEES，如例子 21-3 所示。

例子 21-3　创建一个表 EMPLOYEES

```
SQL> create table employees
  2  (employee id   number(6) not null,
  3   emp_name      varchar2(20),
  21   email         varchar2(25) not null,
  5   phone number  varchar2(20),
  6   hiredate      date    not null);

表已创建。
```

现在我们已经成功创建了一个表，在使用序列号前我们还有一点说明，即两个伪列的使用，一个为 currval，该伪列提供序列的当前值，一个为 nextval，该伪列提供下一个序列号的值。下面演示如何使用序列号自动生成员工号，如例子 21-4 所示，向表中插入两行数据。

例子 21-4　向表 EMPLOYEES 中插入两行数据

```
SQL>  insert into employees
  2   values
  3   (emp seq.nextval,'tom','susu@yahoo.com',13988383756,sysdate);

已创建 1 行。
SQL> insert into employees
  2  values
  3  (emp seq.nextval,'larry','larry@yahoo.com',139832182756,sysdate);

已创建 1 行。
```

此时，我们向 EMPLOYEES 表中插入了两行数据，并且员工号是通过序列号产生的，下面查看员工信息，以认识序列号产生的员工号，如例子 21-5 所示。

例子 21-5　查看员工信息来认识序列号产生的员工号

```
SQL> select *
  2  from employees;

EMPLOYEE_ID EMP_NAME    EMAIL     PHONE_NUMBER        HIREDATE
-----------------------------------------------------------------------
1000         tom        susu@yahoo.com      13988383756           06-8 月 -11
1001         larry        larry@yahoo.com     139832182756          06-8 月
-11
```

我们看到 EMPLOYEE_ID 依次为 1000 和 1001，因为序列号 EMP_SEQ 从 1000 开始，步进为 1，所以每使用一次 EMP_SEQ 序列号就增 1。上面在插入数据时我们使用序列号连续插入两行数据，所以出现的员工号是连续的。

注意，序列号是不会逆转的，如果用户添加了一条记录，而后删除该记录，此时序列号已经递增了，下次再使用序列号产生员工号时，会出现不连续现象。

如我们删除一条记录然后插入一条记录，如例子 21-6 所示。

例子 21-6　先删除一条记录再插入一条记录

```
SQL> delete from employees
  2  where employee_id=1001;

已删除 1 行。
SQL>  insert into employees
  2   values
  3   (emp_seq.nextval,'asaf','asaf@yahoo.com',13983872756,sysdate);

已创建 1 行。
```

此时，我们已经成功插入一条记录，而该记录使用了 EMP_SEQ 序列号产生员工号，我们看这个员工号的值，如例子 21-7 所示。

例子 21-7　再次执行查询 EMPLOYEES 表的全部数据

```
SQL> select *
  2  from employees;

EMPLOYEE_ID EMP_NAME      EMAIL         PHONE_NUMBER        HIREDATE
---------------------------------------- -------------------- -----------
 1000 tom                susu@yahoo.com            13988383756          06-8 月 -11
1002 asaf                asaf@yahoo.com            13983872756          06-8 月 -11
```

此时出现了不连续的员工号，主要是因为我们删除了一个员工记录，而序列号又只增不减造成的。但是无论如何使用序列号保持了员工号的唯一性。

在任何时候，我们都可以使用 curryal 伪列查询当前序列号的值，使用 nextval 查询序列号的下一个值，如例子 21-8 所示，此时使用续表 DUAL。

例子 21-8　使用 currval 伪列查询当前序列号的值

```
SQL>select emp_seq.currval,emp_seq.nextval
  2* from dual

   CURRVAL    NEXTVAL
---------------------- ------------------
      1002       1003
```

21.3　修改序列号

随着业务的变化，某些时候需要对已经定义的序列号进行修改，Oracle 允许使用 ALTER SEQUENCE 语句完成对序列号的修改。在给出修改示例前，我们先看看数据字典 USER_SEQUENCE 的属性信息，以了解修改序列号的那些参数，如例子 21-9 所示。

例子 21-9　查看数据字典 USER_SEQUENCE 的属性信息

```
SQL> desc user_sequences;
 名称                                              是否为空? 类型
```

```
     -----------------------------------------------------             --------
-------------
     SEQUENCE_NAME                                NOT NULL VARCHAR2(30)
     MIN VALUE                                             NUMBER
     MAX VALUE                                            NUMBER
     INCREMENT_BY                                  NOT NULL NUMBER
     CYCLE_FLAG                                            VARCHAR2(1)
     ORDER FLAG                                            VARCHAR2(1)
     CACHE SIZE                                     NOT NULL NUMBER
     LAST_NUMBER                                   NOT NULL NUMBER
```

在计划修改参数 CACHE_SIZE 和 INCREMENT_BY，在修改之前，先查询当前的序列号 EMP_SEQ 的相关属性信息，如例子 21-10 所示。

例子 21-10　查询当前的序列号 EMP_SEQ 的相关属性信息

```
SQL> select cache_size,increment_by,cycle_flag
  2  from user sequences
  3* where sequence name like 'EMP%'

CACHE_SIZE INCREMENT_BY CYCLE_FLAG
---------------------- -------------- --------------------------
       20            1        N
```

我们看到序列号 EMP_SEQ 的 CACHE_SIZE 为 20，而 INCREMENT_BY 的值为 1，现在修改序列号 EMP_SEQ 的参数设置，如例子 21-11 所示。

例子 21-11　修改序列号 EMP_SEQ 的参数设置

```
SQL> alter sequence emp_seq
  2  increment by 2
  3  cache 30;

序列已更改。
```

我们已经成功修改了序列号 EMP_SEQ 的参数设置，将参数 INCREMENT_BY 改为 2，而将 CACHE_SIZE 设置为 30。我继续使用数据字典 USER_SEQUENCES 查看修改后的参数，如例子 21-12 所示。

例子 21-12　使用数据字典 USER_SEQUENCES 查看修改后的参数

```
SQL> select cache size,increment by,cycle flag
  2  from user_sequences
  3  where sequence_name like 'EMP%';

CACHE SIZE INCREMENT BY CYCLE FLAG
---------------------- -------------- --------------------------
       30            2        N
```

此时，再向表 EMPLOYEES 中插入一行数据，则序列号步进为 2，因为当前的序列号值为 1002，所以 nextval 为 1004。我们插入一行数据，如例子 21-13 所示。

例子 21-13　向表 EMPLOYEES 插入一行数据

```
SQL> insert into employees
  2  values
  3  (emp_seq.nextval,'susu','asaf@yahoo.com',13983872756,sysdate);

已创建 1 行。
```

此时成功插入一行数据，再使用例子 21-14 查看插入记录的员工号。

例子 21-14　查看插入记录的员工号

```
SQL> select *
  2  from employees;

EMPLOYEE_ID EMP_NAME      EMAIL       PHONE_NUMBER        HIREDATE
----------- ---------- ------------- ------------------- ----------
1000         tom         susu@yahoo.com      13988383756      06-8 月 -13
1002         asaf        asaf@yahoo.com      13983872756      06-8 月 -13
1004        susu       asaf@yahoo.com      13983872756      06-8 月 -13
```

显然此时的 EMPLOYEE_ID 为 1004，在当前值 1002 基础上步进为 2。因为我们修改了序列号 EMP_SEQ 的参数设置，并且当前的序列号在内存中会放置 30 个值。

我们查看一下该序列号的下一个值，如例子 21-15 所示。

例子 21-15　查看一下该序列号的值

```
SQL> select emp_seq.nextval
  2  from dual;

   NEXTVAL
--------------------
     1006
```

显然这个值和我们的步进参数（INCREMENT_BY）是一致的。

最后我们给出修改序列号的语句格式，如下所示。

```
ALTER SEQUENCE sequence name
[INCREMENT BY n]
[{MAXVALUE n|NOMAXVALUE}]
[{MINVALUE n|NOMINVALUE}]
[{CACHE n|NOCACHE}]
[{CYCLE n|NOCYCLE}]
```

下面解释一下各参数的含义：

- INCREMENT BY n：修改序列号每次执行的增长幅度。
- MAXVALUE n|NOMAXVALUE：修改序列号的最大值|不设置最大值上限。
- MINVALUE n|NOMINVALUE：修改序列号的最小值|不设置最小值下限。
- CACHE n|NOCACHE：修改序列号在内存预分配并保存在内存中的个数。
- CYCLE n|NOCYCLE：修改序列号使得序列号达到最大值或最小值后可以继续产生序列号。



修改序列号的用户必须拥有必要的权限，不能修改 START WITH 参数，ALTER SEQUENCE 对于之前产生的序列号没影响，只影响之后产生的序列号。

21.4　删除序列号

当不需要一个序列号时，可以使用 DROP SEQUENCE 指令删除该序列号，下面我们删除序列号 EMP_SEQ，如例子 21-16 所示。

例子 21-16　删除序列号 EMP_SEQ

```
SQL> drop sequence emp_seq;

序列已删除。
```

显示序列已删除，我们使用数据字典 USER_SEQUENCES 来验证是否成功删除序列号 EMP_SEQ，如例子 21-17 所示。

例子 21-17　验证是否成功删除序列号 EMP_SEQ

```
SQL> select sequence_name,increment_by,cache_size
  2  from user_sequences
  3  where sequence_name = 'EMP_SEQ';

未选定行
```

显然数据字典中没有符合条件的记录，说明已经成功删除序列号 EMP_SEQ，下面如果再向表 EMPLOYEES 中插入一行记录，并使用序列号则产生错误，如例子 21-18 所示。

例子 21-18　向表 EMPLOYEES 中插入一行记录验证是否删除序列号

```
SQL> insert into employees
  2  values
  3  (emp_seq.nextval,'susu','asaf@yahoo.com',13983872756,sysdate);
(emp_seq.nextval,'susu','asaf@yahoo.com',13983872756,sysdate)
 *
第 3 行出现错误:
ORA-02289: 序列不存在
```

因为序列对象 EMP_SEQ 已经删除了，所以无法使用序列对象，从而无法实现使用序列号的插入操作。

21.5　什么是同义词

同义词是 Oracle 数据库中对象的别名，我们知道在 Oracle 数据库中对象包括表、视图、物化视图、触发器、序列号、函数、过程以及 Java 对象等，Oracle 可以为这些对象创建同义词。

使用同义词的主要目的是方便用户访问属于其他用户的数据库对象，或出于安全目的，因为

同义词名字具有随机性，没有限制，这样就隐藏了创建同义词的原始对象的信息。同义词可以是公有的，也可以是私有的。顾名思义，公有同义词是任何用户都可以使用的，而私有同义词只有指定的用户可以使用，没有授权其他用户无法访问某一个用户的私有同义词。

下面通过例子分析同义词的优点，如例子 21-19 所示。

例子 21-19　在 SYSTEM 用户下查看 SCOTT 用户的 DEPT 表信息

```
SQL> conn system/oracle
已连接。
SQL> select *
  2  from dept;
from dept
     *
第 2 行出现错误:
ORA-00942: 表或视图不存在
```

显然在 SYSTEM 用户模式下，无法识别 SCOTT 用户的 DEPT，而必须在表 DEPT 前使用 SCOTT 用户模式，说明该表属于 SCOTT 用户，如例子 21-20 所示。

例子 21-20　通过指定表 DEPT 的模式名 SCOTT 查看表信息

```
SQL> conn system/oracle
已连接。
SQL> select *
  2  from scott.dept;

   DEPTNO DNAME          LOC
------------------- ---------------------- --------------
       210 OPERATIONS     BOSTON
       10 ACCOUNTING    NEW YORK
       20 RESEARCH      DALLAS
       30 SALES          CHICAGO
```

可见，使用模式名确实实现了查询，但是却对使用者“暴露”了表 DEPT 的模式信息。而使用别名则可以避免这个问题。下面介绍如何创建同义词。

21.6　创建公有同义词

公有同义词对数据库的所有用户有效，一般是某种应用的所有者创建如过程或程序包的公有同义词，以便其他用户可以使用它们。我们先给出例子说明如何创建公有同义词，如例子 21-21 所示。

例子 21-21　创建公有同义词

```
SQL> create public synonym department for scott.dept;

同义词已创建。
```

在上面成功创建同义词后，就可以使用公有同义词 department 了，如例子 21-22 所示。

例子 21-22　使用公有同义词查询数据

```
SQL> conn system/oracle
已连接。
SQL> select *
  2  from department;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        40 OPERATIONS     BOSTON
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
```

可见使用公有同义词避免了例子 21-19 所示的问题，此时任何用户使用公有同义词 department 都可以访问 SCOTT 用户的 DEPT 表了。

为了更有说服力，我们使用 SCOTT 用户登录，看是否可以使用该同义词，如例子 21-23 所示。

例子 21-23　使用 SCOTT 用户验证使用公有同义词

```
SQL> conn scott/oracle
已连接。
SQL> select *
  2  from department;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        40 OPERATIONS     BOSTON
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
```

现在使用公有同义词，在 SCOTT 用户模式获得需要的数据，其实读者也可以通过其他用户自行验证。



创建公有同义词的用户必须具有 CREATE PUBLIC SYNONYM 的权利，当其他用户使用该公有同义词时，只有具有相应的权限才可以对同义词对应的表执行诸如 DELETE、UPDATE 等操作。

21.7　创建私有同义词

私有同义词和公有同义词相对，私有同义词只对创建它的用户有效，而其他用户无法使用。如例子 21-24 所示。

例子 21-24　创建私有同义词

```
SQL> create synonym d for scott.dept;

同义词已创建。
```

此时，我们成功创建一个私有同义词 d，该同义词代表 SCOTT 用户的对象即表 DEPT。下面在 SYSTEM 用户模式下，使用同义词 d，如例子 21-25 所示。

例子 21-25　验证使用私有同义词

```
SQL> conn system/oracle
已连接。
SQL> select *
  2  from d;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        40 OPERATIONS     BOSTON
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
```

再使用 SCOTT 用户登录数据库，也试图使用私有同义词 d，如例子 21-26 所示。

例子 21-26　在 SCOTT 用户模式下验证私有同义词 d

```
SQL> conn scott/oracle
已连接。
SQL> select *
  2  from d;
from d
     *
第 2 行出现错误:
ORA-009212: 表或视图不存在
```

我们看到私有同义词 d 在 SCOTT 用户模式下无法使用。



创建私有同义词和创建公有同义词的唯一区别就是是否使用 PUBLIC 关键字，如果不使用 PUBLIC 关键字说明是私有同义词，如果使用 PUBLIC 关键字说明是公有同义词。

21.8　删除同义词

删除同义词的指令很简单，使用 DROP 关键字，但是删除公有同义词时必须使用 PUBLIC 关键字，而删除私有同义词时就不需要。我们在 SYSTEM 用户下创建了一个公有同义词 deparment 和一个私有同义词 d，下面我们删除这两个同义词。分别如例子 21-27 和例子 21-28 所示。

例子 21-27　删除公有同义词 department

```
SQL> drop public synonym department;

同义词已删除。
```

例子 21-28　删除私有同义词 d

```
SQL> conn system/oracle
已连接。
```

```
SQL> drop synonym d;

同义词已删除。
```

21.9　切换用户模式

如果当前在一个用户模式下，而需要使用另一个用户模式下的对象，则可以在当前用户模式下切换用户，如当前用户是 SYSTEM，如例子 21-29 所示。

例子 21-29　使用 SYSTEM 模式登录数据库

```
SQL> conn system/oracle
已连接。
```

下面使用例子 21-30 切换到 SCOTT 用户模式，直接使用表 DEPT 查询数据，如例子 21-30 所示。

例子 21-30　切换 SYSTEM 用户模式到 SCOTT 用户模式，并查询表 DEPT 中的数据

```
SQL> alter session set current schema =scott;

会话已更改。

SQL> select *
  2  from dept;

    DEPTNO DNAME          LOC
------------------- ------------------------ -------------
        40 OPERATIONS     BOSTON
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
```



在使用用户模式切换时，当前用户必须拥有对切换后的用户对象的访问权，诸如 SELECT、UPDATE 等权利。

21.10　本章小结

本章我们介绍了两个很有用的 Oracle 对象序列号和同义词，通过本章的学习读者可以把握 Oracle 引入序列号的初衷，以及如何创建和使用序列号。序列号的维护主要包括删除和修改序列号。而同义词确实方便了用户对其他用户的对象的使用，使用公有同义词使得任何用户都可以操作某个数据库对象，如果处于安全考虑则可以使用私有同义词或者使用授权的方式，将用户创建的同义词授权给信任的用户。

第 22 章

RMAN备份与恢复数据库

RMAN 是 Oracle 提供的实用程序 Recovery Manager，即恢复管理器，使用 RMAN 可以轻松实现数据库的所有备份任务，如备份整个数据库、特定的表空间或者数据文件，RMAN 是 Oracle 提供的一个更加智能和自动化的备份恢复工具，具有许多新的特性，如实现增量备份、备份文件的差错检验等，本章将详细介绍 RMAN 技术的每一个细节。

22.1　RMAN概述

RMAN 在数据库服务器的帮助下实现数据库文件、控制文件以及数据库文件和控制文件的映像副本、归档日志文件、数据库服务器参数文件的备份。RMAN 也允许使用脚本文件实现数据的备份与恢复，而且这些脚本保存在数据库内，而不需要编写基于 OS 的脚本文件。RMAN 备份的文件自动保存在一个系统指定的目录下，文件的名称也由 RMAN 自己维护，当实现数据恢复操作时，恢复指令简洁，RMAN 自动寻找需要的文件实现数据恢复。减少了在传统的导出导入程序中人为错误的发生。本节我们将详细介绍 RMAN 备份与恢复的技术细节，包括 RMAN 的优点、RMAN 的系统结构、快闪恢复区、使用 RMAN 脚本、RMAN 参数配置、恢复目录的概念，以及使用 RMAN 实现数据备份和数据恢复。

22.2　RMAN的独特之处

如果读者使用过 EXP/IMP 以及 EXPDP/IMPDP 工具，应该很好理解使用 RMAN 带来的好处。Oracle 每次技术的演进都是使得其功能更强大，操作更简单，更加满足生产数据库的要求。相对“古老”的备份技术，使用 RMAN 的优点如下所示。

- 支持增量备份：在传统的备份工具（如 EXP 或 EXPDP）中，只能实现一个完整备份而不能增量备份，RMAN 采用被备份级别实现增量备份，在一个完整备份的基础上，采用增量备份，和传统的备份方式相比，这样可以减少备份的数据量。
- 自动管理备份文件：RMAN 备份的数据是 RMAN 自动管理的，包括文件名字、备份文件存储目录，以及识别最近的备份文件，搜索恢复时需要的表空间、模式或数据文件等备份文件。
- 自动化备份与恢复：在备份和恢复操作时，使用简单的指令就可以实现备份与恢复，且执

行过程完全由 RMAN 自己维护。

- 不产生重做信息：与用户管理的联机备份不同，使用 RMAN 的联机备份不产生重做信息。
- 恢复目录：RMAN 的自动化备份与恢复功能应该归功于恢复目录的使用，RMAN 直接在其中保存了备份和恢复脚本。
- 支持映像复制：使用 RMAN 也可以实现映像复制，映像是以操作系统上的文件格式存在，这种复制方式类似于用户管理的脱机备份方式。
- 新块的比较特性：这是 RMAN 支持增量备份的基础，这种特性使得在备份时，跳过数据文件中从未使用过的数据块的备份，备份数据量的减少直接导致了备份存储空间需求和备份时间的减少。
- 备份的数据文件压缩处理：RMAN 提供一个参数，说明是否对备份文件进行压缩，压缩的备份文件以二进制文件格式存在，可以减少备份文件的存储空间。
- 备份文件有效性检查功能：这种功能验证备份的文件是否可用，在恢复前往往需要验证备份文件的有效性。

22.3　RMAN系统架构详解

Oracle 的 RMAN 工具使用会话建立客户端到数据库服务器的连接，用户首先需要启动 RMAN 可执行程序，然后建立客户端与服务器端的会话连接，用户通过 RMAN 的客户端进行 RMAN 操作，执行备份与恢复指令，这些指令在服务器端的服务器进程中执行，而服务器进程完成实际的磁盘读写操作。下面，为了完成数据库的备份与恢复操作，我们详细介绍 RMAN 的系统结构组成。

- RMAN 可执行程序：它是一个客户端工具，用来启动与数据库服务器的连接，从而实现备份与恢复的各种操作。
- RMAN 客户端：一旦建立了与数据库服务器的会话连接，RMAN 可执行程序就创建一个客户端，通过客户端完成与数据库服务器之间的通信，完成各种备份与恢复操作的指令。RMAN 客户端可以连接通过 Oracle Net 连接到可访问的任何主机上。
- 服务器进程：在 RMAN 建立了与数据库服务器的会话连接后，在数据库服务器端启动一个后台进程，它执行 RMAN 客户端发出的各种数据恢复与备份指令，并完成实际的磁盘或磁带设备的读写任务。
- RMAN 信息库：RMAN 信息库记录了 RMAN 的一些信息，如备份的数据文件及副本的目录、归档的重做日志备份文件和副本、表空间和数据文件，以及备份或恢复的脚本和 RMAN 的配置信息。默认使用数据库服务器的控制文件记录这些信息，读者可以通过转储的控制文件发现这些信息，如使用 ALTER DATABASE BACKUP CONTROL FILE TO TRACE。
- 恢复目录：记录 RMAN 信息库的信息。但是恢复目录需要事先配置，信息库既可以存储在数据库的控制文件中，也可以存储在恢复目录中。在 Oracle 中默认先将 RMAN 信息库写入控制文件，如果存在恢复目录则需要继续写到恢复目录。使用控制文件的不足是控制文件中记录 RMAN 信息库的空间有限，当空间不足时可能被覆盖掉。所以 Oracle 建议创建单独的恢复目录。这样也可以更好地发挥 RMAN 提供的新特性。



图 22-1 为 RMAN 的系统结构图，其实也可以理解为一个备份或恢复过程的信息流示意图，RMAN 可执行程序启动并建立与数据库服务器的会话连接，客户端发出备份指令，而数据库服务器端的服务器后台进程执行指令完成磁盘读写操作，并将备份信息记录在 RMAN 信息库中，RMAN 信息库可以保存在数据库服务器端的控制文件中，如果使用恢复目录，RMAN 信息库同样会自动保存在恢复目录中，实际上发送到 RMAN 恢复目录的元数据是从控制文件同步来的。

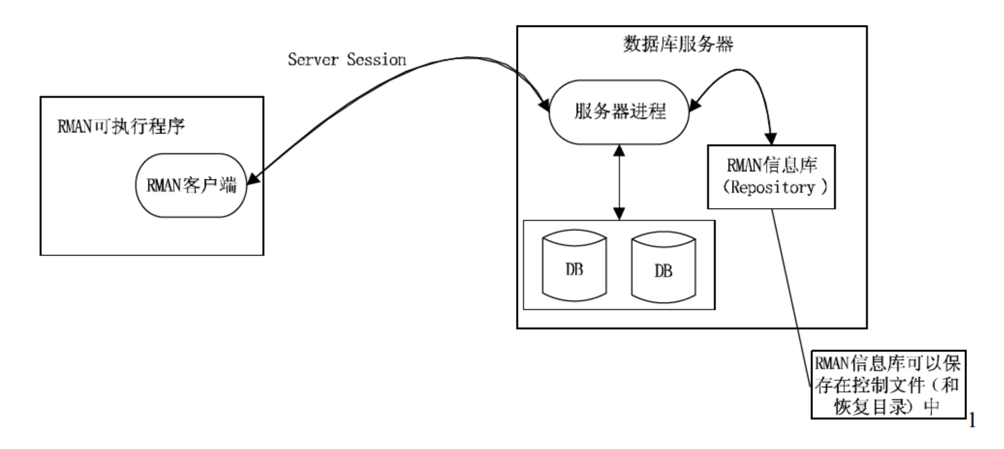


图 22-1 RMAN 的系统结构组成

22.4 快闪恢复区（flash recovery area）

快闪恢复区是存储与备份和恢复数据文件以及相关信息的存储区。快闪恢复区保存了每个数据文件的备份、增量备份、控制文件备份以及归档重做日志备份，Oracle 也允许在快闪恢复区中保存联机重做日志的冗余副本以及当前控制文件的冗余副本，还有 Oracle 中闪回特性中的闪回日志也保存在快闪恢复区中。

在使用 RMAN 实现数据库的备份与恢复时，配置的快闪恢复区就是 RMAN 存储所有与备份相关的文件存储区，而此时的文件名不需要用户干预，Oracle 使用 OMF 创建备份文件的文件名，文件名称格式可以指定。

使用快闪恢复区的优点是，实现了备份文件的自动管理，使得备份与恢复数据库更简单（指令更简洁），并且可以集中管理磁盘空间。要求恢复区的空间足够大，以容纳备份的数据。我们通过例子查看快闪恢复区的两个参数信息。

22.4.1 修改快闪恢复区大小

在快闪恢复区需要设置两个动态参数。一个是 DB_RECOVERY_FILE_DEST_SIZE，该参数用于设置快闪恢复区的最大容量；一个是 DB_RECOVERY_FILE_DEST，该参数设置快闪恢复区在

操作系统磁盘空间上的位置。可以通过两种方式来设置快闪恢复区的参数。一种方法是通过在初始化参数文件 init.ora 文件中设置这两个参数。另一种方法是通过数据库指令 ALTER SYSTEM 在运行的数据库上动态地设置它们。下面演示如何动态设置快闪恢复区的参数。

首先查看当前数据库的快闪恢复区参数，使用 SHOW PARAMETER 指令。

例子 22-1　查看快闪恢复区的参数信息

```
SQL> show parameter db recovery file dest;

NAME                                TYPE       VALUE
----------------------------------------- ------------------------------
db recovery file dest string   /u01/app/oracle/flash recovery  area
db_recovery_file_dest_size          big integer 2G
```

从中可以看到快闪恢复区在磁盘上的目录和快闪恢复区的空间大小，我们备份的整个数据库，以及控制文件都保存在该快闪恢复区中，该区域中的文件由 Oracle 自己维护，一旦需要恢复数据库时，只需要使用简单的指令就可以恢复数据库，RMAN 工具会自动寻找存储在快闪恢复区中的备份文件完成恢复。



快闪恢复区的参数可以动态更改，如可以在数据库运行期间改变快闪恢复区的大小，以及改变快闪恢复区在磁盘上的存储目录。

下面我们演示如何动态修改快闪恢复区的大小，如例子 22-2 所示。

例子 22-2　修改快闪恢复区的参数

```
SQL> alter system set
  2  db_recovery_file_dest_size=2g;

系统已更改。
```

快闪恢复区的参数除了在运行库上动态更改，或者在 init.ora 文件或 SPFILE 文件中设置，也可以使用 OEM 工具的 DATABASE CONTROL 配置快闪恢复区。

为了以后演示方便，我们仍将快闪恢复区的目录设置为其默认目录，大小仍为 2G。

如果不需要快闪恢复区可以将参数 DB_RECOVERY_FILE_DEST 的值设置为空格，使得快闪恢复区不存在存储目录。

当使用了快闪恢复区后，可以通过数据字典 v$recovery_file_dest 来查看快闪恢复区的空间使用情况，以及文件数量，如例子 22-3 所示。

例子 22-3　查看快闪恢复区的位置以及空间使用信息

```
SQL> col name for a30
SQL> set line 100
SQL>select name,space limit,space used,number of files
  2* from v$recovery file dest

NAME                         SPACE_LIMIT SPACE_USED NUMBER_OF_FILES
--------------------------------------------------------------------------
```

```
    /u01/app/oracle/flash_recovery_area    214741936419    2264719592            9
```

上述输出说明当前数据库的快闪恢复区的空间 SPACE_LIMIT 为 2G，已经使用了 SPACE_USED 为 216MB，当前恢复区中的文件数为 9。NAME 的值说明快闪恢复区的操作系统目录，该目录为/u01/app/oracle/flash_recovery_area。

22.4.2 解决快闪恢复区的空间不足问题

那么如果快闪恢复区的空间不足该如何处理呢，有 3 种方法，一是增加恢复区磁盘空间，但这受当前磁盘空间的限制，二是删除没用的备份文件或将备份文件复制到磁带设备，三是删除当前的恢复区，重新设置新的快闪恢复区。

- 增加磁盘空间，我们讲过可以使用 ALTER SYSTEM 指令动态设置快闪恢复区的空间大小，如例子 22-4 所示。

例子 22-4 重新设置快闪恢复区的空间大小

```
SQL> alter system set
  2  db recovery file dest size=4g;

系统已更改。
```

- 使用 CROSSCHECK 和 DELETE OBSOLETE 指令删除不需要的文件。或者使用 DELETE EXPIRED 指令删除那些不需要的备份文件。或者使用 RMAN 的 BACKUP RECOVERY AREA 指令将恢复区中的文件复制到磁带中。
- 删除当前的快闪恢复区，重新设置。

```
SQL> alter system set db_recovery_file_dest =' /u01/backup/newflasharea'
```



当向快闪恢复区添加新文件时，Oracle 会自动更新文件列表，发现符合删除条件的备份文件，这些文件包括不符合保留策略的文件，复制到磁带的过渡文件，而重做日志文件和控制文件任何时候都不会被删除。

闪回恢复区的空间使用如何，我们可以使用 v$flash_recovery_file_usage 来查询。

例子 22-5 查询闪回恢复区的空间使用情况

```
SQL>                                                                    select
file type,percent space used,percent space reclaimable,number of files
  2  from v$flash_recovery_area_usage;

FILE TYPE    PERCENT SPACE USED  PERCENT SPACE RECLAIMABLE  NUMBER OF FILES
-----------------------------------------------------------------------
CONTROLFILE              0                      0              0
ONLINELOG                0                      0              0
ARCHIVELOG             .01                      0              1
BACKUPPIECE              10.53                    5.31           19
IMAGECOPY                0                      0              0
FLASHBACKLOG             0                      0              0
```

```
6 rows selected.
```

从查询结果可知，在闪回恢复区中保存的文件类型有控制文件、归档日志文件、在线日志文件、备份片、影像备份文件以及闪回日志（和闪回相关的历史记录）。

22.5　建立RMAN到数据库的连接

此时，我们介绍了 RMAN 的基本功能、系统组成，以及使用 RMAN 实现备份与恢复的快闪恢复区，本节讲如何使用 RMAN 建立到数据库服务器的连接。下面通过例子说明连接到数据库服务器的方法。

例子 22-6　使用数据库用户名和密码登录 RMAN

```
RMAN> connect target system/oracle

connected to target database: TEST (DBID=205919193796)
```

例子 22-6 说明首先在操作系统环境下输入 RMAN 指令，启动 RMAN 可执行程序，而后使用 connect target 指令使数据库用户名和密码建立与数据库服务器的会话连接，通常情况下我们创建一个特定的用户用于 RMAN 连接，该用户要求具有 SYSDBA 权限。如下所示，我们创建一个用户 rman，密码为 oracle，具有 DBA 角色。

例子 22-7　创建用户 rman 并赋予一系列权限和资源

```
SQL> create user rman identified by oracle;

User created.

SQL> grant resource,connect ,dba to rman;

Grant succeeded.
```

接下来使用该用户连接 RMAN。

例子 22-8　使用 rman 用户连接 RMAN

```
[oracle@ocm1 ~]$ rman target rman/oracle

Recovery Manager: Release 11.2.0.1.0 - Production on Tue Sep 13 17:25:52 2011

Copyright (c) 1992, 2005, Oracle.  All rights reserved.

connected to target database: TEST (DBID=205919193796)
```

也可以使用操作系统认证连接到 RMAN。

例子 22-9　使用操作系统认证连接到 RMAN

```
[oracle@ocm1 ~]$ rman target /
```

```
Recovery Manager: Release 11.2.0.1.0 - Production on Tue Sep 13 17:27:57 2011

Copyright (c) 1992, 2005, Oracle.  All rights reserved.

connected to target database: TEST (DBID=205919193796)
```

22.6 RMAN的相关概念与配置参数

进一步讨论使用 RMAN 进行更多类型的备份前，我们有必要说明几个 RMAN 概念，这些概念也多次出现在备份输出过程中，理解这些概念是使用 RMAN 备份的基础，如下所示。

- 备份集：备份集是一个逻辑数据集合，由一个或多个 RMAN 的备份片组成，备份片是 RMAN 格式的操作系统文件，包含数据文件、控制文件或者归档日志文件。默认情况下，在执行 RMAN 的备份时，将产生备份文件的备份集，备份集只有 RMAN 可以识别，所以在恢复时必须使用 RMAN 来访问备份集实现恢复。备份集是 RMAN 默认的备份文件类型，备份集就是备份片的逻辑组合。一般一个通道生成一个备份集，如果设置了控制文件自动备份，则控制文件的备份文件单独生成一个备份集。控制文件的备份集以操作系统块作为最小单位，数据文件备份集以数据库块作为最小单位，所以它们不能放在一个备份集合中。
- 通道：RMAN 是通过与数据库服务器的会话建立连接，通道代表这个连接，它指定了备份或恢复数据库的备份集所在的设备，如磁盘或磁带。
- 映像复制：映像复制是数据库文件的操作系统文件的一个备份，就如使用操作系统的 COPY 指令备份的文件一样，一个数据文件生成一个映像文件副本，整个复制过程是 RMAN 进行数据块的复制过程，RMAN 会检测每一个数据块是否出现损坏，不需要将表空间设置成为 begin backup，镜像副本中包含使用过的数据块，也包含从未使用过的数据块。生成镜像副本的好处在于恢复速度相对备份集来说，更快一些。

使用 RMAN 将默认创建备份集，它是数据集的一个逻辑数据结构，也可以设置备份类型为 COPY，使得使用 RMAN 的任何备份不产生备份集，而产生映像复制。如下所示。

```
RMAN> CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO COPY
```

也可以使用 BACKUP AS COPY 指令实现备份数据的映像复制，如下所示。

例子 22-10　映像复制整个数据库

```
RMAN> BACKUP AS COPY DATABASE
```

例子 22-11　映像复制单个表空间

```
RMAN> BACKUP AS COPY TABLESPACE USERS
```

例子 22-12　映像复制整个数据库中的一个数据文件

```
RMAN> BACKUP AS COPY DATAFILE 3
```

参数 DATAFILE 后的数值表示数据文件 ID，它与数据字典 DBA_DATA_FILES 中的 FILE_ID 参数一致。

接下来我们分析一下 RMAN 的配置参数，首先登录 RMAN 之后，使用 SHOW ALL 指令显示当前的所有 RMAN 参数，如例子 22-13 所示。

例子 22-13　查看 RMAN 的配置参数

```
RMAN> show all;

RMAN 配置参数为:
CONFIGURE RETENTION POLICY TO REDUNDANCY 1; # default
CONFIGURE BACKUP OPTIMIZATION OFF; # default
CONFIGURE DEFAULT DEVICE TYPE TO DISK; # default
CONFIGURE CONTROLFILE AUTOBACKUP OFF; # default
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '%F'; # default
CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET; # default
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default
CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default
CONFIGURE MAXSETSIZE TO UNLIMITED; # default
CONFIGURE ENCRYPTION FOR DATABASE OFF; # default
CONFIGURE ENCRYPTION ALGORITHM 'AES128'; # default
CONFIGURE ARCHIVELOG DELETION POLICY TO NONE; # default
CONFIGURE          SNAPSHOT          CONTROLFILE          NAME          TO
'F:\ORACLE\PRODUCT\10.2.0\DB_1\DATABASE\SNCFORCL.ORA'; # default
```

可以根据需要更改上述中的参数，我们先解释部分 RMAN 参数的含义，然后说明如何设置该参数。

- CONFIGURE RETENTION POLICY TO REDUNDANCY 1：该参数说明保留备份的副本数量，如果每天都备份一个数据文件，上述参数 1 说明只保留一个该数据文件的副本，并且保留最新的备份副本。
- CONFIGURE DEFAULT DEVICE TYPE TO DISK：该配置参数说明备份的数据文件默认备份到数据库服务器的磁盘上，该参数可以更改为备份到磁带上，如例子 22-14 所示。

例子 22-14　更改 RMAN 的备份设备类型为磁带

```
RMAN> configure default device type to sbt;

新的 RMAN 配置参数:
CONFIGURE DEFAULT DEVICE TYPE TO 'SBT_TAPE';
已成功存储新的 RMAN 配置参数
释放的通道: ORA_DISK_1
```

这里仅仅是为了说明设备类型的更改方式，读者最好使用如下方式将设备类型恢复为磁盘。

```
RMAN> configure default device type to disk;
```

- CONFIGURE BACKUP OPTIMIZATION OFF：配置备份优化，模式不使用备份优化，使用备份优化的作用是如果已经备份了某个文件的相同版本，则不会再备份该文件，即只保留一份备份文件。可以使用如下方式打开备份优化。

```
RMAN> CONFIGURE BACKUP OPTIMIZATION ON;

新的 RMAN 配置参数:
```

```
CONFIGURE BACKUP OPTIMIZATION ON;
已成功存储新的 RMAN 配置参数
```

- CONFIGURE CONTROLFILE AUTOBACKUP OFF：配置模式不启动控制文件的自动备份，更改方式就是将 OFF 设置为 ON，修改指令如下所示。

```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;
```

一旦更改之后，如果数据库结构发生变化或者在备份数据库过程中，控制文件会自动再备份到指定的目录下。

- CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET：该参数说明 RMAN 在备份和恢复中的并行度，在执行备份或恢复时，通道数量越多，则任务执行时间越短，备份文件类型为备份集，当然备份文件类型也可以是映像副本（COPY）。修改并行数以及备份文件类型的指令如下所示。

```
RMAN> CONFIGURE DEVICE TYPE DISK PARALLELISM 3 BACKUP TYPE TO COPY;
```

22.7　RMAN备份控制文件

RMAN 可以读单独备份控制文件，如果没有启用快闪恢复区则使用 FORMAT 参数指定控制文件的备份目录，如果启用了快闪恢复区，RMAN 会自动将控制文件复制到快闪恢复区的备份集中（BACKUPSET 目录下），下面通过两个例子演示如何使用备份控制文件的指令和整个备份过程。

例子 22-15　在没有启用快闪恢复区时备份控制文件

```
RMAN> backup current controlfile format
2> 'f:\pump\backup_ctl_%u.dbf';
启动 backup 于 29-8 月 -09
使用通道 ORA_DISK_1
通道 ORA_DISK_1: 启动全部数据文件备份集
通道 ORA_DISK_1: 正在指定备份集中的数据文件
备份集中包括当前控制文件
通道 ORA_DISK_1: 正在启动段 1 于 29-9 月 -09
通道 ORA_DISK_1: 已完成段 1 于 29-9 月 -09
段句柄=F:\PUMP\BACKUP_CTL_0CKNTU1G.DBF 标记=TAG20090829T171527 注释
=NONE
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:02
完成 backup 于 29-8 月 -09
```



在本章的备份例子中多次使用替换变量%U，它的作用是产生唯一的备份文件名。因为没有使用快闪恢复区,所以在执行控制文件恢复时，DBA 必须知道备份目录，显然这增加了 DBA 的工作负担。

下面是使用快闪恢复区时的备份控制文件方式。

例子 22-16　在启用快闪恢复区时备份控制文件

```
RMAN> backup current controlfile ;
```

```
启动 backup 于 29-8 月 -09
使用通道 ORA_DISK_1
通道 ORA DISK 1: 启动全部数据文件备份集
通道 ORA DISK 1: 正在指定备份集中的数据文件
备份集中包括当前控制文件
通道 ORA_DISK_1: 正在启动段 1 于 29-9 月 -09
通道 ORA DISK 1: 已完成段 1 于 29-9 月 -09
段句柄
=F:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\BACKUPSET\2009 09
29\O1_MF_NCNNF_TAG20090829T17179_59KWK66T_.B
KP 标记=TAG20090829T17179 注释=NONE
通道 ORA DISK 1: 备份集已完成, 经过时间:00:00:02
完成 backup 于 29-9 月 -09
```

使用快闪恢复区的好处就是 Oracle 自动管理文件的备份目录，文件命名使用 OMF，DBA 也不需要记住这个目录，在恢复时同样不需要记住该目录，RMAN 将使用 RMAN 信息库记录的信息找到备份的文件集。

控制文件记录了数据库的物理文件组成等重要信息，一旦数据库结构发生变化（如创建了新的表空间），控制文件就会更新，此时最好备份该数据文件，对 DBA 是一个很麻烦的事情。Oracle 的 RMAN 支持控制文件的自动备份，使得在数据库结构发生变化，控制文件更新时自动备份控制文件，在执行 BACKUP 备份数据库时也会在备份的最后阶段备份控制文件。如下演示如何设置控制文件自动备份，并且备份到指定目录。

例子 22-17　配置控制文件备份的磁盘类型和备份目录

```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO
 '/u01/backup/%F';

new RMAN configuration parameters:
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO
'/u01/backup/%F';
new RMAN configuration parameters are successfully stored
```

例子 22-18　配置控制文件自动备份。

```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;

new RMAN configuration parameters:
CONFIGURE CONTROLFILE AUTOBACKUP ON;
new RMAN configuration parameters are successfully stored
```

下面做个测试，删除一个表空间，此时的数据库结构发生变化，由于启动了控制文件自动备份，在目录/u01/backup 下会自动备份控制文件。

例子 22-19　删除一个表空间。

```
SQL> drop tablespace test including contents and datafiles;

Tablespace dropped.
```

查看目录/u01/backup 下是否有备份的控制文件。

例子 22-20　查看目录/u01/backup 下是否有备份的控制文件。

```
SQL> ! ls -al /u01/backup
total 98190619
drwxr-x-x  2 oracle oinstall     4096 Sep 14 12:24 .
drwxr-xr-x  6 oracle oinstall     4096 Sep 10 20:44 ..
-rw-r-----  1 oracle oinstall  7143424 Sep 14 12:24 c-205919193796-20110914-00
```

在目录/u01/backup 下备份了控制文件，备份的文件名格式为 c-DBID-控制文件备份日期-序列号，每天的备份数量不能超过 256 个。

22.8　RMAN实现脱机备份

下面通过实例说明如何实现 RMAN 的脱机备份，要实现脱机备份首先需要使用 RMAN 登录到数据库服务器，关闭数据库然后启动数据库到 MOUNT 状态，再执行 BACKUP DATABASE 指令备份整个数据库。在备份之前我们查看或设置两个重要参数。

首先查看下设备类型是磁带还是磁盘。

例子 22-21　查看下设备类型是磁带还是磁盘

```
RMAN> backup as compressed backupset database;

Starting backup at 14-SEP-11
//使用控制文件代替恢复目录存储备份元数据
using target database control file instead of recovery catalog
//分配通道，并行度为 3 所以分配了 3 个通道，设备类型为磁盘
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=156 devtype=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: sid=155 devtype=DISK
allocated channel: ORA_DISK_3
channel ORA_DISK_3: sid=154 devtype=DISK
//通道 1 启动压缩的数据文件备份集备份，分配了数据文件 1
channel ORA_DISK_1: starting compressed full datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
input datafile fno=00001 name=/u01/app/oracle/oradata/TEST/system01.dbf
channel ORA_DISK_1: starting piece 1 at 14-SEP-11
//通道 2 启动压缩的数据文件备份集备份，分配了数据文件 3 和 4
channel ORA_DISK_2: starting compressed full datafile backupset
channel ORA_DISK_2: specifying datafile(s) in backupset
input datafile fno=00003 name=/u01/app/oracle/oradata/TEST/sysaux01.dbf
input datafile fno=00004 name=/u01/app/oracle/oradata/TEST/users01.dbf
channel ORA_DISK_2: starting piece 1 at 14-SEP-11
//通道 2 启动压缩的数据文件备份集备份，分配了数据文件 5 和 2
channel ORA_DISK_3: starting compressed full datafile backupset
channel ORA_DISK_3: specifying datafile(s) in backupset
input datafile fno=00005 name=/u01/app/oracle/oradata/TEST/example01.dbf
input datafile fno=00002 name=/u01/app/oracle/oradata/TEST/undotbs01.dbf
channel ORA_DISK_3: starting piece 1 at 14-SEP-11
//通道 3 备份过程，备份集信息，消耗时间
```

```
    channel ORA DISK 3: finished piece 1 at 14-SEP-11
    piecehandle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_14/
o1_mf_nnndf_TAG20
    110914T124651 770dfwm19 .bkp tag=TAG20110914T124651 comment=NONE
    channel ORA DISK 3: backup set complete, elapsed time: 00:01:15
    //通道 2 备份过程，备份集信息，消耗时间
    channel ORA_DISK_2: finished piece 1 at 14-SEP-11
    piecehandle=/u01/app/oracle/flash recovery area/TEST/backupset/2011 09 14/
o1 mf nnndf TAG20
    110914T124651 770dfw0y .bkp tag=TAG20110914T124651 comment=NONE
    channel ORA_DISK_2: backup set complete, elapsed time: 00:01:23
    //通道 1 备份过程，备份集信息，消耗时间
    channel ORA DISK 1: finished piece 1 at 14-SEP-11
    piecehandle=/u01/app/oracle/flash recovery area/TEST/backupset/2011 09 14/
o1 mf nnndf TAG20
    110914T124651_770dfvpw_.bkp tag=TAG20110914T124651 comment=NONE
    channel ORA DISK 1: backup set complete, elapsed time: 00:01:419
    Finished backup at 14-SEP-11
    //启动控制文件和动态参数文件的自动备份，二者放在一个备份集中，存储目录为/u01/backup。
    Starting Control File and SPFILE Autobackup at 14-SEP-11
    piece handle=/u01/backup/c-205919193796-20110914-01 comment=NONE
    Finished Control File and SPFILE Autobackup at 14-SEP-11
```

当备份整个数据库时，RMAN 将自动备份控制文件和服务器参数文件，其实这取决于 RMAN 的 CONFIGURE CONTROLFILE AUTOBACKUP ，设置该参数值为 ON 使得在使用 RMAN 执行任何备份指令时，自动备份控制文件和 SPFILE 文件。

最后打开数据库。

例子 22-22　打开数据库

```
    RMAN> sql 'alter database open';

    sql statement: alter database open
```

此时，使用 RMAN 完成了整个数据库的脱机备份。

22.9　RMAN联机备份

本节我们学习 RMAN 的联机备份，在使用联机备份前必须满足一定的条件，比如使得数据库处于归档模式等，然后我们通过例子详解如何备份整个数据库，表空间以及数据文件。

22.9.1　联机备份前的准备工作

在使用 RMAN 进行联机备份前，必须设置快闪恢复区，将 DB_RECOVERY_FILE_DEST 参数指定的目录作为归档重做日志备份的默认位置，并且将快闪恢复区的尺寸设置得足够大，此时使用快闪恢复区作为备份的存储路径，在 Oracle 9i 以及之前的版本还需要将 LOG_ARCHIVE_START 参数的值设置为 TRUE，如下所示。

例子 22-23　将参数 LOG_ARCHIVE_START 设置为 TRUE

```
SQL> alter system set log archive start = true scope =spfile;

系统已更改。
```

但是在 Oracle 10g 及以上版本中，一旦启动归档模式，归档进程自动启动。

在进行联机备份前都要求将数据库置于归档模式，因为处于联机备份的数据库中要备份的所有数据文件头中的 SCN 被锁定，但是此时在数据库中的数据文件的表依然可以被访问，并执行 DML 操作，但是这些修改的数据不能写入数据文件，重做日志进程将这些变化的数据全部写到重做日志文件，如果备份的时间很长，而且在这期间产生了大量的变化数据，重做日志会切换从而将这些变化的数据写到归档日志文件中。

显然，RMAN 的联机备份使得数据库可以不间断业务的运行，而且通过 RMAN 可以备份整个数据库，一个表空间或者一个数据文件，可以灵活选择备份的粒度，对于超大型数据库，如果备份整个数据库是相当耗时的，而生产数据库中往往只需要备份某个重要的表空间或数据文件，如只读表空间，非在线的 UNDO 表空间都可以不备份，而只备份联机事务的表空间。联机备份时因为启用了归档模式，不会丢失数据更新。在介质故障时，可以实现数据库的全恢复。但是请注意，必须小心保存或备份归档日志，因为一旦它丢失或损坏就无法实现数据库完全恢复，只能实现不完全恢复。

最后需要将数据库置为归档模式，其操作步骤是先关闭数据库再启动数据库到 MOUNT 状态，然后使用 ALTER DATABASE ARCHIVELOG 将数据库设置为归档模式。打开数据库就可以进行 RMAN 联机热备份了。

例子 22-24　将数据库设置为归档模式

```
SQL> shutdown immediate;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup
ORACLE instance started.

Total System Global Area  2195212672 bytes
Fixed Size                 12119992 bytes
Variable Size             7549901919 bytes
Database Buffers          2055201996 bytes
Redo Buffers                2973696 bytes
Database mounted.
SQL> alter database archivelog;

Database altered.
```

为了验证修改结果，最好使用例子 22-25 所示查看当前数据库的归档模式。

例子 22-25　查看数据库的归档模式

```
SQL> archive log list;
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            USE_DB_RECOVERY_FILE_DEST
```

```
    Oldest online log sequence     1
    Next log sequence to archive   2
    Current log sequence           2
```

RMAN 使用快闪恢复区作为备份文件的存储区，并且将数据库的归档日志也存放在快闪恢复区中，文件的名称格式为 OMF 文件格式。

下面依次通过例子说明如何使用 RMAN 联机备份整个数据库，备份一个表空间，备份一个数据文件以及备份当前的控制文件。

22.9.2　联机备份整个数据库

下面，我们演示如何使用 BACKUP DATABASE 联机备份整个数据库，目的是备份数据文件，同时也备份归档日志文件。在备份完成归档日志文件后，将已经备份的归档日志文件从存储目录中删除，这样即备份了归档日志文件，同时清除了归档空间，如例子 22-26 所示。

例子 22-26　使用 RMAN 备份整个数据库

```
    RMAN> backup as compressed backupset database plus archivelog delete all input;

    Starting backup at 14-SEP-11
    current log archived
    using target database control file instead of recovery catalog
    allocated channel: ORA_DISK_1
    channel ORA_DISK_1: sid=142 devtype=DISK
    allocated channel: ORA_DISK_2
    channel ORA_DISK_2: sid=141 devtype=DISK
    allocated channel: ORA_DISK_3
    channel ORA_DISK_3: sid=140 devtype=DISK
    channel ORA_DISK_1: starting compressed archive log backupset
    channel ORA_DISK_1: specifying archive log(s) in backup set
    input archive log thread=1 sequence=1 recid=7 stamp=761192191947
    input archive log thread=1 sequence=2 recid=19 stamp=7611967912
    channel ORA_DISK_1: starting piece 1 at 14-SEP-11
    channel ORA_DISK_2: starting compressed archive log backupset
    channel ORA_DISK_2: specifying archive log(s) in backup set
    input archive log thread=1 sequence=3 recid=9 stamp=76119619160
    channel ORA_DISK_2: starting piece 1 at 14-SEP-11
    channel ORA_DISK_3: starting compressed archive log backupset
    channel ORA_DISK_3: specifying archive log(s) in backup set
    input archive log thread=1 sequence=3 recid=6 stamp=761741519
    channel ORA_DISK_3: starting piece 1 at 14-SEP-11
    channel ORA_DISK_3: finished piece 1 at 14-SEP-11
    piecehandle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_14/
o1_mf_annnn_TAG20
    110914T220242_771f03v6_.bkp tag=TAG20110914T220242 comment=NONE
    channel ORA_DISK_3: backup set complete, elapsed time: 00:00:01
    channel ORA_DISK_3: deleting archive log(s)
    archive log
    filename=/u01/app/oracle/flash_recovery_area/TEST/archivelog/2011_09_13/o1
_mf_1_3_76xkbh05_.a
    rc recid=6 stamp=761741519
```

```
    channel ORA DISK 1: finished piece 1 at 14-SEP-11
    piecehandle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_14/
o1_mf_annnn_TAG20
    110914T220242 771f03bq .bkp tag=TAG20110914T220242 comment=NONE
    channel ORA DISK 1: backup set complete, elapsed time: 00:00:05
    channel ORA_DISK_1: deleting archive log(s)
    archive log
    filename=/u01/app/oracle/flash recovery area/TEST/archivelog/2011 09 14/o1
 mf 1 1 7706m3x1 .
    arc recid=7 stamp=761192191947
    archive log
    filename=/u01/app/oracle/flash_recovery_area/TEST/archivelog/2011_09_14/o1
 mf 1 2 771dr7z6 .ar
    c recid=19 stamp=7611967912
    channel ORA DISK 2: finished piece 1 at 14-SEP-11
    piecehandle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_14/
o1 mf annnn TAG20
    110914T220242 771f03go .bkp tag=TAG20110914T220242 comment=NONE
    channel ORA DISK 2: backup set complete, elapsed time: 00:00:05
    channel ORA_DISK_2: deleting archive log(s)
    archive log
    filename=/u01/app/oracle/flash recovery area/TEST/archivelog/2011 09 14/o1
 mf 1 3 771dzzcs .ar
    c recid=9 stamp=76119619160
    Finished backup at 14-SEP-11
```

上面这部分使用了 3 个通道备份了归档日志文件，并且在备份完成后删除归档目录下的相应归档文件。接下来这部分使用 3 个通道备份数据文件。

```
    Starting backup at 14-SEP-11
    using channel ORA DISK 1
    using channel ORA DISK 2
    using channel ORA_DISK_3
    channel ORA_DISK_1: starting compressed full datafile backupset
    channel ORA DISK 1: specifying datafile(s) in backupset
    input datafile fno=00001 name=/u01/app/oracle/oradata/TEST/system01.dbf
    channel ORA_DISK_1: starting piece 1 at 14-SEP-11
    channel ORA_DISK_2: starting compressed full datafile backupset
    channel ORA DISK 2: specifying datafile(s) in backupset
    input datafile fno=00003 name=/u01/app/oracle/oradata/TEST/sysaux01.dbf
    input datafile fno=00004 name=/u01/app/oracle/oradata/TEST/users01.dbf
    channel ORA_DISK_2: starting piece 1 at 14-SEP-11
    channel ORA_DISK_3: starting compressed full datafile backupset
    channel ORA DISK 3: specifying datafile(s) in backupset
    input datafile fno=00005 name=/u01/app/oracle/oradata/TEST/example01.dbf
    input datafile fno=00002 name=/u01/app/oracle/oradata/TEST/undotbs01.dbf
    channel ORA_DISK_3: starting piece 1 at 14-SEP-11
    channel ORA DISK 3: finished piece 1 at 14-SEP-11
    piecehandle=/u01/app/oracle/flash recovery area/TEST/backupset/2011 09 14/
o1 mf nnndf TAG20
    110914T2202419_771f0m4h_.bkp tag=TAG20110914T2202419 comment=NONE
    channel ORA_DISK_3: backup set complete, elapsed time: 00:01:06
    channel ORA_DISK_2: finished piece 1 at 14-SEP-11
```

```
    piecehandle=/u01/app/oracle/flash recovery area/TEST/backupset/2011 09 14/
o1_mf_nnndf_TAG20
    110914T2202419_771f0c6j_.bkp tag=TAG20110914T2202419 comment=NONE
    channel ORA DISK 2: backup set complete, elapsed time: 00:01:09
    channel ORA DISK 1: finished piece 1 at 14-SEP-11
    piecehandle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_14/
o1_mf_nnndf_TAG20
    110914T2202419 771f0g0w .bkp tag=TAG20110914T2202419 comment=NONE
    channel ORA DISK 1: backup set complete, elapsed time: 00:01:44
    Finished backup at 14-SEP-11
```

在上述备份完成后，Oracle 会自动发生一次日志切换，继续备份剩余的归档日志，因为只有一个归档日志文件，虽然启动了 3 个通道，但是只使用了通道 1 作为备份通道。

```
    Starting backup at 14-SEP-11
    current log archived
    using channel ORA DISK 1
    using channel ORA DISK 2
    using channel ORA_DISK_3
    channel ORA_DISK_1: starting compressed archive log backupset
    channel ORA DISK 1: specifying archive log(s) in backup set
    input archive log thread=1 sequence=4 recid=10 stamp=76119619273
    channel ORA DISK 1: starting piece 1 at 14-SEP-11
    channel ORA_DISK_1: finished piece 1 at 14-SEP-11
    piecehandle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_14/
o1 mf annnn TAG20
    110914T220433 771f3m6j .bkp tag=TAG20110914T220433 comment=NONE
    channel ORA_DISK_1: backup set complete, elapsed time: 00:00:02
    channel ORA_DISK_1: deleting archive log(s)
    archive log
    filename=/u01/app/oracle/flash recovery area/TEST/archivelog/2011 09 14/o1
 mf 1 4 771f3kcj .ar
    c recid=10 stamp=76119619273
    Finished backup at 14-SEP-11
```

启动控制文件和动态参数文件的自动备份。

```
    Starting Control File and SPFILE Autobackup at 14-SEP-11
    piece handle=/u01/backup/c-205919193796-20110914-02 comment=NONE
    Finished Control File and SPFILE Autobackup at 14-SEP-11
```

此时，我们成功备份了整个数据库，备份后数据库文件名称和存储目录都由 RMAN 自己维护，不需要 DBA 干预，这极大地减少了备份和恢复中出错的几率。



在备份整个数据库时，其实就是备份了数据文件，其中包含了当前的控制文件和参数文件。而重做日志文件或归档日志文件不是联机状态数据库全备份的内容，所以使用联机热备份的数据库在数据恢复时需要 recover 数据库，即将联机备份开始到故障点之间的所有提交的数据重新写入数据文件。

在联机备份时，也手工指定多个通道并将数据文件分布到不同的通道上去。整个指令集合如下所示。

```
RMAN>run {
2> allocate channel ch1 device type disk format '/u01/backup/ch1_%U';
3> allocate channel ch2 device type disk format '/u01/backup/ch2_%U';
4> backup as backupset
5>(datafile 1,4 channel ch1)
6>(datafile 2,3,5 channel ch2);
7> sql 'alter system archive log current';}
```

将所有的备份片放在同一个目录下面。Format 参数可以在指定通道的时候指定，也可以在 backup 命令中指定，如例子 22-27 所示。

例子 22-27　Format 参数可以在指定通道时和 backup 命令中指定

```
RMAN>run {
2> allocate channel ch1 device type disk;
3> allocate channel ch2 device type disk;
4> backup as backupset format '/u01/backup/%U'
5>(datafile 1,4 channel ch1)
6>(datafile 2,3,5 channel ch2);
7> sql 'alter system archive log current';}
```

上述 RMAN 备份的流程图如图 22-2 所示。

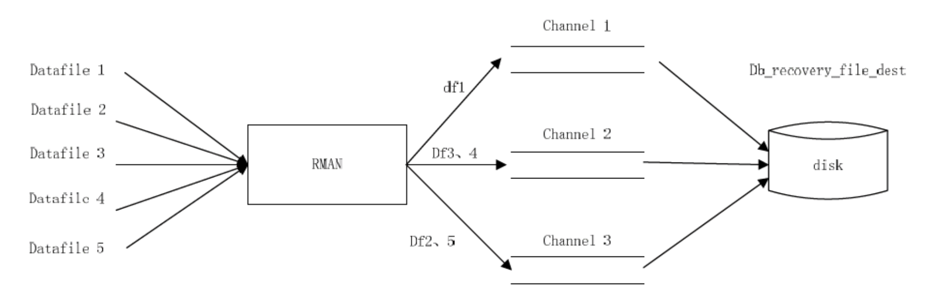


图 22-2　RMAN 备份的流程图

RMAN 备份数据文件，使用默认的 3 个通道，自动将数据文件大致平均地分配到三个通道上，并备份至快闪恢复区。

如果不指定 DATAFILE 的分配，那么 RMAN 会根据数据文件的大小大致进行分配，尽量保持两个通道分配的平均。手工分配通道以后会将默认的通道覆盖掉，默认通道的分配取决于参数 PARALLELISM。

22.9.3　联机备份一个表空间

在很多情况下，我们需要经常备份一个表空间，比如 SYSTEM 表空间，此时就可以使用 BACKUP TABLESPACE 联机备份一个表空间，如例子 22-28 所示。

例子 22-28　使用 RMAN 备份表空间

```
RMAN> backup tablespace users;
```

```
    Starting backup at 14-SEP-11
    using channel ORA_DISK_1
    using channel ORA_DISK_2
    using channel ORA_DISK_3
    channel ORA_DISK_1: starting full datafile backupset
    channel ORA_DISK_1: specifying datafile(s) in backupset
    input datafile fno=00004 name=/u01/app/oracle/oradata/TEST/users01.dbf
    channel ORA_DISK_1: starting piece 1 at 14-SEP-11
    channel ORA_DISK_1: finished piece 1 at 14-SEP-11
    piecehandle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_14/
o1_mf_nnndf_TAG20
    110914T2211945_771fy5rc_.bkp tag=TAG20110914T2211945 comment=NONE
    channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
    Finished backup at 14-SEP-11

    Starting Control File and SPFILE Autobackup at 14-SEP-11
    piece handle=/u01/backup/c-205919193796-20110914-03 comment=NONE
    Finished Control File and SPFILE Autobackup at 14-SEP-11
```

在上述例子中，我们备份了表空间 USERS，备份的文件类型为备份集，在第一次使用 RMAN 备份生成备份集时，在 RMAN 的快闪恢复区中会自动创建一个目录 BACKUPSET，又会根据日期创建新的目录，将一天中的备份集放在一个根据日期创建的目录下。

在备份表空间 USERS 时，虽然分配了 3 个通道，因为只有一个数据文件，只在通道 1 上创建了一个备份集，这个备份集包括一个备份片。控制文件和 spfile 单独备份到了一个位置，生成了文件。

为了减少占用的存储空间，可以使用压缩备份，具体指令如下所示。

```
    RMAN>backup as compressed backupset tablespace users;
```

压缩比大概是 5：1，如果表空间的数据文件很大，使用压缩备份确实可以极大减少存储空间的压力。

22.9.4　联机备份一个数据文件

在备份一个数据文件前，通常查看当前数据库中的所有数据文件，以便于选择需要备份的数据文件。下面是备份特定的数据文件的例子，备份文件类型为备份集，备份集的目录和名称由 format 决定，其中通道生成的备份文件的名称和格式如下所示。

```
    %c：备份片的副本数
    %d：数据库名称
    %D：位于该月的第几天
    %M：位于该年的第几个月
    %n：数据库名称，向右填补到最大 19 个字符
    %u：一个 19 个字符的名称，代表备份集和创建时间
    %p：该备份集的备份片号，从 1 开始到创建的文件数
    %U：一个唯一的名字%u_%p_%c
    %s：备份集的编号
    %t：备份集的时间戳
    %T：年月日格式（YYYY-MM-DD）
```

例子 22-29　使用 RMAN 备份数据文件

```
    RMAN> backup as backupset datafile 1 format '/u01/backup/datafile 1 %U';

    Starting backup at 14-SEP-11
    using channel ORA DISK 1
    using channel ORA DISK 2
    using channel ORA_DISK_3
    channel ORA_DISK_1: starting full datafile backupset
    channel ORA DISK 1: specifying datafile(s) in backupset
    input datafile fno=00001 name=/u01/app/oracle/oradata/TEST/system01.dbf
    channel ORA DISK 1: starting piece 1 at 14-SEP-11
    channel ORA_DISK_1: finished piece 1 at 14-SEP-11
    piece   handle=/u01/backup/datafile_1_0pmmidau_1_1   tag=TAG20110914T222742
comment=NONE
    channel ORA DISK 1: backup set complete, elapsed time: 00:00:45
    Finished backup at 14-SEP-11

    Starting Control File and SPFILE Autobackup at 14-SEP-11
    piece handle=/u01/backup/c-205919193796-20110914-04 comment=NONE
    Finished Control File and SPFILE Autobackup at 14-SEP-11
```

如果 RMAN 没有配置自动备份控制文件，则经常联机备份控制文件是好习惯，一旦数据库结构发生了变化，如新建了表空间、添加或删除了数据文件等。

例子 22-30　使用 RMAN 备份控制文件

```
    RMAN> backup current controlfile;

    Starting backup at 14-SEP-11
    using channel ORA_DISK_1
    using channel ORA_DISK_2
    using channel ORA DISK 3
    channel ORA DISK 1: starting full datafile backupset
    channel ORA DISK 1: specifying datafile(s) in backupset
    including current control file in backupset
    channel ORA_DISK_1: starting piece 1 at 14-SEP-11
    channel ORA DISK 1: finished piece 1 at 14-SEP-11
    piece
    handle=/u01/app/oracle/flash recovery area/TEST/backupset/2011 09 14/o1 mf
_ncnnf_TAG2011091
    4T223211 771gqddo .bkp tag=TAG20110914T223211 comment=NONE
    channel ORA DISK 1: backup set complete, elapsed time: 00:00:01
    Finished backup at 14-SEP-11
```

22.9.5　RMAN 备份坏块处理方式

默认情况下，RMAN 在备份时总是会检查数据块是否发生物理损坏，为了加快备份速度而不希望 RMAN 进行数据块的物理检查时，可以关闭这个选项。

```
    RMAN>backup nochecksum tablespace users tag='weekly_backup';
```

默认情况下，RMAN 不会检查数据块是否发生逻辑损坏，backup 时可以启用逻辑损坏检查。

```
RMAN>backup check  logical tablespace users;
```

RMAN 进行备份时，只要发现新的坏块，就立即停止备份。如果发现的坏块是上次已经发现的，则继续备份。我们可以设置 maxcorrupt 参数来通知 RMAN，只有当发现的坏块个数超过指定的数量时，才停止备份。

这是一个迫不得已才使用的参数，尽量不要使用。

```
RMAN> run {
2> set maxcorrupt for datafile 2,4 to 10;
3> backup database;
4> }
```

数据文件 2、4 出现的新的坏块超过 10 的时候则停止备份。显然我们不希望出现坏块的情况，此时最后使用以前的 RMAN 的备份恢复这个坏块，修复后再备份该数据文件。

22.10　RMAN的增量备份

在使用 BACKUP DATABASE 时，都是全库备份，显然每次这样的备份时很耗费时间也占用磁盘空间，而 RMAN 的增量备份具有很多优势，它只备份自上次全备份以来变化的数据。显然增量备份比全库备份要快，因为增量备份的数据量明显减少（相对于全库备份而言）。

这里需要解释两个级别的增量备份，级别 0 的增量备份和级别 1 的增量备份。其中级别 0 的增量备份与全库备份相同。而级别 1 备份执行的是差异备份，即对级别 0 备份后变化的数据做备份。显然级别 0 备份是级别 1 备份的数据基础。

例子 22-31　使用 RMAN 实现增量备份的级别 0 备份

```
RMAN> backup incremental level 0 database;

启动 backup 于 01-9 月 -09
使用通道 ORA DISK 1
通道 ORA DISK 1: 启动增量级别 0 数据文件备份集
通道 ORA DISK 1: 正在指定备份集中的数据文件
输入数据文件 fno=00002
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF
输入数据文件 fno=00001
 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF
输入数据文件 fno=00003
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF
输入数据文件 fno=00005
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF
输入数据文件 fno=00004 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF
通道 ORA_DISK_1: 正在启动段 1 于 01-9 月 -09
通道 ORA_DISK_1: 已完成段 1 于 01-9 月 -09
段句柄
=F:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\BACKUPSET\2009 09 01\O1
_MF_NNND0_TAG20090901T171114_59SS9MDP_.B
KP 标记=TAG20090901T171114 注释=NONE
```

```
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:02:56
通道 ORA_DISK_1: 启动增量级别 0 数据文件备份集
通道 ORA_DISK_1: 正在指定备份集中的数据文件
备份集中包括当前控制文件
在备份集中包含当前的 SPFILE
通道 ORA_DISK_1: 正在启动段 1 于 01-9月 -09
通道 ORA_DISK_1: 已完成段 1 于 01-9月 -09
段句柄
=F:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\2009_09_01\O1
_MF_NCSN0_TAG20090901T171114_59SSH4CQ_.B
KP 标记=TAG20090901T171114 注释=NONE
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:03
完成 backup 于 01-9月 -09
```

例子 22-31 的全库备份集是增量备份的数据基础，在使用以后使用增量备份的级别 1 的第一次备份时，将变化的数据记录到增量备份集。而当第二次使用级别 1 增量备份时，只备份自上次增量备份以来的所有变化的数据。这种级别 1 的增量备份称为差异备份。还有一种级别 1 的增量备份即累积备份，每次实现增量备份时，它总是备份自级别 0 备份以来所有变化的数据。显然差异增量备份会有多个备份文件，而累积增量备份只有一个备份文件，所以使用累积增量备份可以减少数据库的恢复时间。下面是使用级别 1 增量备份的例子，它采用级别 1 的差异增量备份。

例子 22-32　使用 RMAN 实现增量备份的级别 1 备份

```
RMAN> backup incremental level 1 database;

启动 backup 于 01-9月 -09
使用通道 ORA_DISK_1
通道 ORA_DISK_1: 启动增量级别 1 数据文件备份集
通道 ORA_DISK_1: 正在指定备份集中的数据文件
输入数据文件 fno=00002
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF
输入数据文件 fno=00001
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF
输入数据文件 fno=00003
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF
输入数据文件 fno=00005
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF
输入数据文件 fno=00004 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF
略过数据文件 00004, 因为它未更改
通道 ORA_DISK_1: 正在启动段 1 于 01-9月 -09
通道 ORA_DISK_1: 已完成段 1 于 01-9月 -09
段句柄
=F:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\2009_09_01\O1
_MF_NNND1_TAG20090901T1720019_59SST9QG_.B
KP 标记=TAG20090901T1720019 注释=NONE
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:55
通道 ORA_DISK_1: 启动增量级别 1 数据文件备份集
通道 ORA_DISK_1: 正在指定备份集中的数据文件
备份集中包括当前控制文件
在备份集中包含当前的 SPFILE
通道 ORA_DISK_1: 正在启动段 1 于 01-9月 -09
```

```
通道 ORA_DISK_1: 已完成段 1 于 01-9月 -09
段句柄
=F:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\2009_09_01\O1
 MF_NCSN1_TAG20090901T172019_59SSW2PG_.B
KP 标记=TAG20090901T172019 注释=NONE
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:03
完成 backup 于 01-9月 -09
```

此时，我们完成了使用 RMAN 实现增量备份的所有步骤。在生产数据库中，读者可以使用操作系统工具编写脚本文件，从而实现增量备份的自动化。

22.11　快速增量备份

在 22.10 节我们介绍如何实现增量备份，使用增量备份大大减少了全库备份的时间，同时也节约了存储空间，但是使用增量备份必须扫描整个数据文件，因为无论在上次增量备份数据库是否发生变化，都要进行一次全扫描来确认是否有变化的数据，为了避免这种情况的发生，Oracle 提供了快速增量备份的方案，其原理是将数据库中发生变化的数据块位置记录在一个更改跟踪文件中，这样在下次实现增量备份时就可以通过该文件来备份变化的数据，这样就减少了全库扫描的时间。

在启动了块更改跟踪特性后，会启动一个后台进程 CTWR 负责将变化的数据块的位置写入定义的块跟踪文件。如例子 22-33 所示，启动快跟踪特性。

例子 22-33　启动快跟踪特性

```
SQL> alter database enable block change tracking
  2  using file 'e:/oracle/product/10.2.0/oradata/chtrack.log';

数据库已更改。
```

此时我们将块跟踪文件保存在和数据文件相同的目录下，文件名为 chtrack.log。如果该文件丢失或者损坏会造成数据库无法启动，需要禁用块跟踪特性后方可启动成功启动数据库。

下面通过数据字典视图 v$block_change_tracking 查询启动。

例子 22-34　查询是否启动快跟踪特性

```
SQL> col filename for a50
SQL> select filename,status,bytes from v$block_change_tracking

FILENAME                                    STATUS          BYTES
------------------------------------------- ---------------------------- ----------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\CHTRACK.LOG       ENABLED      11599872
```

通过视图查询清楚地表明已经启动块跟踪特性，记录数据块变化的文件存储在系统的数据文件目录中，即 E:\ORACLE\PRODUCT\10.2.0\ORADATA\CHTRACK.LOG。文件大小为 11 599 872B。

在使用块跟踪特性过程中，如果需要重命名或更改跟踪文件的存放位置，可以使用 ALTER DATABASE RENAME FILE 指令实现，但是必须将数据库启动到 MOUNT 状态，如例子 22-35 所示。

例子 22-35　更改块跟踪文件的存储位置

```
SQL> alter database rename file
```

```
  2  'e:/oracle/product/10.2.0/oradata/chtrack.log'
  3  to
  4  'e:/oracle/product/10.2.0/oradata/lszpri/chtrack.log';

数据库已更改。
```

此时，将块跟踪文件从 e:/oracle/product/10.2.0/oradata/chtrack.log 更改到新的目录下，新目录为 e:/oracle/product/10.2.0/oradata/lszpri/chtrack.log。

在不需要时，可以通过如下方式禁用块跟踪特性，如例子 22-36 所示。

例子 22-36　禁用块跟踪特性

```
SQL> alter database disable block change tracking;

数据库已更改。
```

在禁用块跟踪特性后，通过数据字典视图 v$block_change_tracking 确认禁用结果，如下所示。

例子 22-37　查看禁用块跟踪特性结果

```
SQL> select filename,status,bytes from v$block_change_tracking;

FILENAME                                     STATUS        BYTES
-------------------------------- -------------------------- ----------
                                           DISABLED
```

从输出看出，当前的块跟踪特性为 DISABLED，说明禁用成功，此时的数据文件为空。

22.12　在映像副本上应用增量备份

Oracle 11g 能够将增量备份应用到某个镜像副本上，映像副本是 Oracle 数据库对数据库的一个映像备份。

- 周日生成了一个镜像副本。
- 周一进行增量备份，然后将产生的增量备份应用到周日所做的镜像副本上，这时周日的镜像副本中就包含了周一的数据，从而体现了最新的状态。
- 周二的增量备份再应用到镜像副本上，以此类推。

对镜像副本应用增量备份的最大的好处在于加快恢复速度。

例子 22-38　对镜像副本应用增量备份

```
RMAN> run {
2> backup incremental level 1 for recover of copy with tag 'incr copy backup'
database;
3> recover copy of database with tag 'incr_copy_backup';
4> }

Starting backup at 15-SEP-11
using channel ORA_DISK_1
using channel ORA_DISK_2
```

```
using channel ORA DISK 3
no parent backup or copy of datafile 1 found
no parent backup or copy of datafile 3 found
no parent backup or copy of datafile 5 found
no parent backup or copy of datafile 2 found
no parent backup or copy of datafile 4 found
channel ORA_DISK_1: starting datafile copy
input datafile fno=00001 name=/u01/app/oracle/oradata/TEST/system01.dbf
channel ORA DISK 2: starting datafile copy
input datafile fno=00003 name=/u01/app/oracle/oradata/TEST/sysaux01.dbf
channel ORA_DISK_3: starting datafile copy
input datafile fno=00005 name=/u01/app/oracle/oradata/TEST/example01.dbf
……
Starting recover at 15-SEP-11
using channel ORA DISK 1
using channel ORA_DISK_2
using channel ORA DISK 3
no copy of datafile 1 found to recover
no copy of datafile 2 found to recover
no copy of datafile 3 found to recover
no copy of datafile 4 found to recover
no copy of datafile 5 found to recover
Finished recover at 15-SEP-11

Starting Control File and SPFILE Autobackup at 15-SEP-11
piece handle=/u01/backup/c-205919193796-20110915-00 comment=NONE
Finished Control File and SPFILE Autobackup at 15-SEP-11
```

第一条语句表示要生成级别为 1 的 tag 值为 incr_copy_backup，针对整个数据库并且应用于增量备份的镜像副本。

（1）第一次执行该程序时，在执行第一条语句时，由于我们没有 0 级备份，因此会生成整个数据库的镜像副本（该副本不是普通的副本，而是可以在其上应用增量备份）。如下所示查询当前的映像副本。

例子 22-39　查询当前的映像副本

```
RMAN> list copy;

specification does not match any archive log in the recovery catalog

List of Datafile Copies
Key     File S Completion Time Ckp SCN    Ckp Time        Name
------- ---- - --------------- ---------- --------------- ----
6            1         A  15-SEP-11               525401           15-SEP-11
/u01/app/oracle/flash_recovery_area/TEST/datafile/o1_mf_system_773zqovx_.dbf
3             2         A  15-SEP-11               525433           15-SEP-11
/u01/app/oracle/flash recovery area/TEST/datafile/o1 mf undotbs1 773ztdh4 .dbf
5             3         A  15-SEP-11               525402           15-SEP-11
/u01/app/oracle/flash recovery area/TEST/datafile/o1 mf sysaux 773zqpdc .dbf
4             4         A  15-SEP-11               525443           15-SEP-11
/u01/app/oracle/flash_recovery_area/TEST/datafile/o1_mf_users_773zv7p1_.dbf
```

```
    2                5         A   15-SEP-11               525403          15-SEP-11
/u01/app/oracle/flash_recovery_area/TEST/datafile/o1_mf_example_773zqqjt_.dbf
```

执行第二条语句的时候，因为没有增量备份，因此不会执行，但是也不会报错。

（2）第二次执行该程序，由于已经有了第一次的 0 级备份，所以会生成一个 1 级别的增量备份，执行第二条语句的时候，会将第一条语句生成的增量备份应用到第一次所生成的镜像副本上。以后的每一次都会生成一个增量备份，并将该生成的增量备份应用到镜像副本上，如果需要恢复，先恢复镜像副本，然后应用最近一次增量备份以来的所有归档日志，就可实现数据库的完全恢复。

每天重复执行上面的 run 模块，就能够达到我们要的如下效果。

- 每天将增量备份添加到镜像副本上。
- 恢复时使用最近的镜像副本+少量的归档日志，恢复速度会很快。

22.13　创建和维护恢复目录

恢复目录保存了 RMAN 信息库的信息，Oracle 推荐使用恢复目录保存 RMAN 信息库，在信息库中保存了数据文件备份集或映像复制，表空间和数据文件信息以及 RMAN 的配置信息，使用恢复目录 RMAN 在一定条件下读取目标库的控制文件来更新恢复目录中保存的关于控制文件、数据文件等信息。

下面演示在不同数据库上创建恢复目录的步骤和方法。

- 创建恢复目录表空间。

为了存储 RMAN 备份元数据，使用 PROD 库的一个特定表空间存储元数据。下面是在 PROD 库上创建一个表空间 rcat_tbs。

例子 22-40　在 PROD 库上创建一个表空间 rcat_tbs

```
[oracle@ocm1 ~]$ sqlplus sys/oracle@prod as sysdba

SQL*Plus: Release 11.2.0.1.0 - Production on Sun Sep 11 15:50:25 2011

Copyright (c) 1992, 2005, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, Oracle Label Security, OLAP and Data Mining options

SQL> create tablespace rcat tbs
  2  datafile '/u01/app/oracle/oradata/PROD/rcat_tbs01.dbf' size 100m;

Tablespace created.
```

- 创建恢复目录用户。

创建用户用于恢复目录，为该用户创建默认表空间用来存储恢复目录，如例子 22-41 所示。

例子 22-41 创建恢复目录用户

```
SQL> create user rcat owner identified by oracle
  2  default tablespace rcat_tbs
  3  temporary tablespace temp;

User created.

SQL> alter user rcat_owner quota unlimited on rcat_tbs;

User altered.
```

以上创建了用户 rcat_owner，该用户使用 rcat_tbs 表空间存储恢复目录。

- 为恢复目录授权。

为了使用新用户，首先对新用户 rcat_owner 进行授权，使得该用户称为恢复目录的拥有者，将 RECOVERY_CATALOG_OWNER 角色赋予用户 rcat_owner。

例子 22-42 为恢复目录用户授权

```
SQL> grant recovery catalog owner to rcat owner;

Grant succeeded.

SQL> grant connect,resource to rcat_owner;

Grant succeeded.
```

- 连接到恢复目录和目标数据库。

要使用恢复目录，必须先连接到恢复目录数据库，如下所示。

例子 22-43 连接到恢复目录数据库

```
D:\>rman catalog rman backup/rman@orcl

恢复管理器: Release 11.1.0.6.0 - Production on 星期四 9月 3 08:52:53 2013

Copyright (c) 1992, 2007, Oracle.  All rights reserved.

连接到恢复目录数据库
```



在笔者电脑上恢复目录数据库和目标数据库在一个数据库上，所以上述方式也同时连接到了目标数据库，如果二者不在同一数据库上，如目标数据库在 leejia 数据库上，则在连接到恢复目录数据库后，再连接到目标数据库，如下所示。

例子 22-44 连接到目标数据库

```
RMAN> connect target system/oracle@leejia

连接到目标数据库: ORCL (DBID=128192901)
```

例子 22-44 中，我们先连接到恢复目录数据库，然后在 RMAN 环境下再连接到名为 LEEJIA 的目标数据库上，也可以使用如下指令一次连接到恢复目录和目标数据库。

例子 22-45 连接到恢复目录数据库和目标数据库

```
RMAN> connect catalog rman_backup/rman@orcl target system/oracle@leejia
```

- 创建恢复目录。

在刚才创建的用户 rman_backup 中创建恢复目录，首先使用用户 rman_backup 登录数据库，然后使用 create catalog 指令创建恢复目录，如例子 22-46 所示。

例子 22-46 创建恢复目录

```
[oracle@ocm1 ~]$ rman catalog rcat_owner/oracle@prod

Recovery Manager: Release 11.2.0.1.0 - Production on Sun Sep 11 15:53:05 2011

Copyright (c) 1992, 2005, Oracle.  All rights reserved.

connected to recovery catalog database

RMAN> create catalog tablespace rcat tbs;

recovery catalog created
```

此时的恢复目录保存在 rcat_tbs 表空间中，该表空间就是在创建 rcat_owner 用户时指定的默认表空间。在不需要恢复目录时，可以使用 DROP CATALOG 指令删除恢复目录，读者可以自己验证。

- 注册目标数据库。

在创建了恢复目录后注册目标数据库，目的是使得恢复目录知道目标数据库的信息，并自动与目标数据库通信获得相关的元数据。要注册目标数据库，必须首先连接到目标数据库。

例子 22-47 在恢复目录中注册目标数据库

```
[oracle@ocm1 ~]$ rman target rman/oracle catalog rcat_owner/oracle@prod

Recovery Manager: Release 11.2.0.1.0 - Production on Sun Sep 11 15:54:59 2011

Copyright (c) 1992, 2005, Oracle.  All rights reserved.

connected to target database: PROD (DBID=15519119122)
connected to recovery catalog database

RMAN> register database;

database registered in recovery catalog
starting full resync of recovery catalog
full resync complete
```

在例子 22-47 中把目标数据库 test 成功注册到恢复目录中，“正在启动全部恢复目录的 resync”的含义是 RMAN 读取目标库的控制文件信息来保存关于数据文件、日志切换等的元数据信息，因为在刚注册时恢复目录没有任何关于目标数据库的注册信息，所以使用同步的方式获取所需信息。其实，在目标数据库结构发生变化后可以使用手工同步的方式，如下所示。

例子 22-48　同步恢复目录

```
RMAN> resync catalog;
```

22.14　RMAN的脚本管理

对于长指令的 RMAN 备份操作 Oracle 提供了脚本语言功能，使得用户正对特定的任务编写备份脚本，然后将脚本存储在恢复目录或存储为文本文件。下面通过例子说明如何创建和使用脚本。

例子 22-49　创建 RMAN 备份脚本

```
RMAN> create script rman_backup{
2> sql 'alter system checkpoint';
3> backup database format
4> '/u01/backup/offline_backup\back_%u.dbf';
5> backup current controlfile format
6> '/u01/backup/offline backup\back ctl %u.dbf';
7> }

已创建脚本 rman_backup
```

注意在创建 RMAN 备份脚本时必须连接到恢复目录和目的数据库，否则不能创建成功。

执行 RMAN 脚本，此时使用 RUN 指令和 EXECUTE SCRIPT 指令执行创建的脚本，其实执行脚本的过程就是执行一系列的 SQL 语句的过程，RMAN 脚本类似于 Windows 中的批处理文件，如例子 22-50 所示。

例子 22-50　执行脚本

```
RMAN> run {execute script rman backup;}

正在执行脚本: rman backup
sql 语句: alter system checkpoint
启动 backup 于 03-9月 -09
.........
```

在介绍完了如何创建脚本以及执行脚本，下面我们介绍如何使用操作系统文件存储 RMAN 指令，并在 RMAN 中直接调用该文件执行 RMAN 命令。

在 Windows 环境下，我们首先创建一个 rman_backup.rcv 文件，如图 22-3 所示。



图 22-3　创建执行 RMAN 指令的操作系统文件

然后将该文件保存在 D 盘的根目录下，如例子 22-51 所示在 RMAN 中调用操作系统文件执行 RMAN 指令。

例子 22-51　调用操作系统文件执行 RMAN 指令

```
D:\>rman catalog rman_backup/rman@orcl target system/oracle@orcl cmdfile
'rman_backup.rcv'

恢复管理器: Release 11.1.0.6.0 - Production on 星期四 9月 3 10:05:45 2009

Copyright (c) 1992, 2007, Oracle. All rights reserved.

连接到目标数据库: ORCL (DBID=1219192901)
连接到恢复目录数据库

RMAN> sql 'alter system checkpoint';
2> backup database format
3> 'f:\offline backup\back %u.dbf';
4> backup current controlfile format
5> 'f:\offline_backup\back_ctl_%u.dbf';
6>
7>
sql 语句: alter system checkpoint

启动 backup 于 03-9月 -09
分配的通道: ORA DISK 1
…………
完成 Control File and SPFILE Autobackup 于 03-9月 -09
恢复管理器完成。
```

为了编辑方便，也可以将存储在恢复目录中的脚本文件转换为操作系统文件，如例子 22-52 所示。

例子 22-52　将脚本文件转换为操作系统文件

```
RMAN> print script rman backup to file 'rman backup.txt';

已将脚本 rman_backup 写入文件 rman_backup.txt
```

22.15　使用RMAN非归档模式下的完全恢复

我们以恢复整个数据库为例说明在非归档模式下使用 RMAN 实现脱机备份的恢复。此时，自脱机备份以来变化的数据可能部分丢失，读者应该还有印象联机重做日志文件是循环使用的，一旦写满一个日志文件将切换到下一个，新的循环开始覆盖掉部分变化的数据。这样的恢复其实是不完全恢复，因为数据库工作在非归档模式下。

首先创造一个模拟环境，数据库处于非归档模式，做数据库脱机备份，并且 RMAN 的参数是使用快闪恢复区作为备份文件存储目录，配置了控制文件自动备份，具体步骤读者自行完成。然后，通过如下故障场景来模拟如何恢复。

22.15.1　控制文件、数据文件以及重做日志文件丢失的恢复

数据文件、控制文件以及重做日志文件全部丢失时很少出现的情况，但是在现实中也确实存在这样的数据库，开发商的程序员不懂得数据库的维护，在部署系统时安装数据库都使用默认设置，这样所有的数据库文件都存放在一个目录下，而一旦磁盘损坏则所有的数据库文件全部丢失。下面为了模拟这个过程，我们先对非归档模式下的数据库做一个全备份。此时需要将数据库启动到 MOUNT 状态。

```
SQL> startup mount;
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                  1218992 bytes
Variable Size              71304784 bytes
Database Buffers          209715200 bytes
Redo Buffers                2973696 bytes
Database mounted.
```

然后使用 RMAN 备份数据库，这个备份集包含了数据文件、控制文件和动态初始化参数文件，并使用快闪恢复区作为备份文件的存储目录。

例子 22-53　使用快闪恢复区作为备份文件的存储目录

```
[oracle@ocm1 ~]$ rman target rman/oracle

Recovery Manager: Release 11.2.0.1.0 - Production on Fri Sep 16 11:32:49 2011

Copyright (c) 882, 2005, Oracle.  All rights reserved.

connected to target database: TEST (DBID=205919193796, not open)

RMAN> backup as compressed backupset database;

Starting backup at 16-SEP-11
using target database control file instead of recovery catalog
…限于篇幅，此处省略备份的部分输出信息…
channel ORA_DISK_1: finished piece 1 at 16-SEP-11
piecehandle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/
```

```
o1 mf nnndf TAG20
    110916T113310_775jvq4f_.bkp tag=TAG20110916T113310 comment=NONE
    channel ORA_DISK_1: backup set complete, elapsed time: 00:02:08
    Finished backup at 16-SEP-11

    Starting Control File and SPFILE Autobackup at 16-SEP-11
    piece handle=/u01/backup/c-2059883796-20110916-00 comment=NONE
    Finished Control File and SPFILE Autobackup at 16-SEP-11
```

在备份了数据库后，为了演示备份后是否有数据丢失，我们创建一个表，并插入部分数据。如下所示。

例子 22-54　创建一个表 test123

```
SQL> create table test123 as select * from dba segments;

Table created.
```

下面模拟控制文件、数据文件，以及重做日志文件的丢失。删除所有这些文件。

```
[oracle@ocm1 ~]$ cd /u01/app/oracle/oradata/TEST
[oracle@ocm1 TEST]$ ls
control01.ctl  example01.dbf  redo03.log    temp01.dbf
control02.ctl  redo01.log     sysaux01.dbf  undotbs01.dbf
control03.ctl  redo02.log     system01.dbf  users01.dbf
[oracle@ocm1 TEST]$ rm -rf *.*
```

此时，如果重新启动数据库会提示无法锁定控制文件，显然这个结果在预料之中，因为我们删除了所有的控制文件，而数据库启动的第二步就是读取控制文件，如下所示。

```
SQL> startup
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                  1218992 bytes
Variable Size              71304784 bytes
Database Buffers          209715200 bytes
Redo Buffers                2973696 bytes
ORA-00205: error in identifying control file, check alert log for more info
```

下面开始恢复过程，因为此时数据库处于 NOMOUNT 状态，首先需要恢复控制文件。因为控制文件是自动备份的，我们使用如下指令恢复控制文件。

例子 22-55　使用如下指令恢复控制文件

```
RMAN> restore controlfile from '/u01/backup/c-2059883796-20110916-00';

Starting restore at 16-SEP-11
using channel ORA_DISK_1

channel ORA DISK 1: restoring control file
channel ORA_DISK_1: restore complete, elapsed time: 00:00:04
output filename=/u01/app/oracle/oradata/TEST/control01.ctl
output filename=/u01/app/oracle/oradata/TEST/control02.ctl
```

```
output filename=/u01/app/oracle/oradata/TEST/control03.ctl
Finished restore at 16-SEP-11
```

此时，恢复了 3 个控制文件，至于控制文件备份集的存储目录以及文件名，通过 RMAN 的配置参数可知。我们通过操作系统指令查看 3 个控制文件是否恢复到原始目录（这个原始位置是参数文件中记录的）。

```
[oracle@ocm1 ~]$ ls /u01/app/oracle/oradata/TEST
control01.ctl  control02.ctl  control03.ctl
```

此时，我们看到成功恢复了 3 个控制文件，但是重做日志文件以及数据文件都没有恢复，下面继续恢复数据库。启动数据库到 MOUNT 状态。

```
SQL> alter database mount;

Database altered.
```

此时，在控制文件中记录了数据文件的恢复位置，以及重做日志文件的位置，数据文件我们通过备份恢复，而重做日志文件我们通过重建的方式创建，此时是新的重做日志文件，日志序列号从 0 开始计数。如下所示，恢复数据文件。

例子 22-56　恢复数据文件

```
RMAN> restore database;

Starting restore at 16-SEP-11
released channel: ORA_DISK_1
Starting implicit crosscheck backup at 16-SEP-11
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=155 devtype=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: sid=154 devtype=DISK
allocated channel: ORA_DISK_3
channel ORA_DISK_3: sid=153 devtype=DISK
Crosschecked 3 objects
………

using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3

channel ORA_DISK_1: starting datafile backupset restore
channel ORA_DISK_1: specifying datafile(s) to restore from backup set
restoring datafile 00002 to /u01/app/oracle/oradata/TEST/undotbs01.dbf
restoring datafile 00005 to /u01/app/oracle/oradata/TEST/example01.dbf
channel ORA_DISK_1: reading from backup piece /u01/app/oracle/flash_recovery_
area/TEST/backupset/2011_09_16/o1_mf_nnndf_TAG20110916T113310_775jvqv3_.bkp
channel ORA_DISK_2: starting datafile backupset restore
channel ORA_DISK_2: specifying datafile(s) to restore from backup set
restoring datafile 00003 to /u01/app/oracle/oradata/TEST/sysaux01.dbf
restoring datafile 00004 to /u01/app/oracle/oradata/TEST/users01.dbf
channel ORA_DISK_2: reading from backup piece
/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/o1_mf_nnndf_
```

```
TAG20110916T1133
    10_775jvqnj_.bkp
    channel ORA_DISK_3: starting datafile backupset restore
    channel ORA_DISK_3: specifying datafile(s) to restore from backup set
    restoring datafile 00001 to /u01/app/oracle/oradata/TEST/system01.dbf
    channel ORA_DISK_3: reading from backup piece
    /u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/o1_mf_nnndf_
TAG20110916T1133
    10_775jvq4f_.bkp
    channel ORA_DISK_2: restored backup piece 1
    piece
    handle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/o1_mf
_nnndf_TAG201109
    16T113310_775jvqnj_.bkp tag=TAG20110916T113310
    channel ORA_DISK_2: restore complete, elapsed time: 00:00:35
    channel ORA_DISK_1: restored backup piece 1
    piece
    handle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/o1_mf
_nnndf_TAG201109
    16T113310_775jvqv3_.bkp tag=TAG20110916T113310
    channel ORA_DISK_1: restore complete, elapsed time: 00:00:38
    channel ORA_DISK_3: restored backup piece 1
    piece
    handle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/o1_mf
_nnndf_TAG201109
    16T113310_775jvq4f_.bkp tag=TAG20110916T113310
    channel ORA_DISK_3: restore complete, elapsed time: 00:01:13
    Finished restore at 16-SEP-11
```

RMAN 从快闪恢复区读取备份集，通过记录的元数据将数据文件恢复到控制文件中记录的位置，下面我们通过操作系统指令验证恢复结果。

```
    [oracle@ocm1 ~]$ ls /u01/app/oracle/oradata/TEST
    control01.ctl  control03.ctl  sysaux01.dbf  undotbs01.dbf
    control02.ctl  example01.dbf  system01.dbf  users01.dbf
```

我们看到所有的数据文件得到恢复，下面恢复（RECOVER）数据库并使用 NOREDO 选项，因为已经丢失了所有重做日志，所以无法使用 REDO 数据了。

例子 22-57　恢复（RECOVER）数据库并使用 NOREDO 选项

```
    RMAN> recover database noredo;

    Starting recover at 16-SEP-11
    using channel ORA_DISK_1
    using channel ORA_DISK_2
    using channel ORA_DISK_3
    Finished recover at 16-SEP-11
```

接下来就是打开数据库了，因为没有了重做日志文件，此时我们使用 RESETLOGS 参数打开数据库会创建新的重做日志文件，日志序列号从 1 开始计数，如下所示。

```
    SQL> alter database open resetlogs;
```

```
Database altered.
```

我们再次通过操作系统指令验证重做日志文件是否恢复，如下所示。

```
[oracle@ocm1 ~]$ ls /u01/app/oracle/oradata/TEST
control01.ctl  example01.dbf  redo03.log    temp01.dbf
control02.ctl  redo01.log     sysaux01.dbf  undotbs01.dbf
control03.ctl  redo02.log     system01.dbf  users01.dbf
```

显然，在当前目录下成功恢复了重做日志文件，这个步骤读者可以不用检验，这里是为了说明更清晰而演示的。因为如果没有重做日志文件数据库根本无法打开。

下面我们查看当前重做日志的序列号如何计数的。

例子 22-58　查看当前重做日志的序列号如何计数的

```
SQL> select group#,sequence#,status from v$log;

    GROUP#  SEQUENCE# STATUS
---------- ---------- ----------------
         1          0 UNUSED
         2          1 CURRENT
         3          0 UNUSED
```

显然，日志组 2 为当前的重组日志组，其日志序列号 sequence#为 1。

读者应该记得，在非归档模式下备份数据库之后，我们创建了一个表 test123。在本案例的恢复中，显然没有更新到该表的创建时，也就是说该表数据会全部丢失。下面在恢复后的数据库中验证该表是否存在。

```
SQL> select count(*) from test123;
select count(*) from test123
                     *
ERROR at line 1:
ORA-00942: table or view does not exist
```

显然，提示表不存在，所以自上次备份以来的所有数据丢失。

22.15.2　只有数据文件丢失的恢复

在非归档模式下，数据文件丢失依然有可能实现完全恢复。我们知道如果用户的重做数据始终没有被覆盖（虽然有日志组的切换），这样日志组中依然记录了备份以来的所有变化的数据，这样依然可以实现完全恢复。

只有数据文件丢失的恢复很简单，使用 RMAN 的两个指令就可以完成，一个是 RESTORE 指令，一个是 RECOVER 指令。下面我们删除当前表空间 USERS 的数据文件。看如何实现完全恢复。

首先，模拟故障删除 USERS 表空间的数据文件。

```
[oracle@ocm1 TEST]$ rm -rf users01.dbf
```

再次启动数据库会提示无法识别或锁定数据文件 users01.dbf。

```
SQL> startup ;
```

```
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                 1218992 bytes
Variable Size             71304784 bytes
Database Buffers         209715200 bytes
Redo Buffers               2973696 bytes
Database mounted.
ORA-01157: cannot identify/lock data file 4 - see DBWR trace file
ORA-01110: data file 4: '/u01/app/oracle/oradata/TEST/users01.dbf'
```

此时，数据库处于 MOUNT 状态，我们使用 RMAN 恢复该数据文件。

例子 22-59　使用 RMAN 恢复 users01.dbf 数据文件

```
RMAN> restore datafile 4;

Starting restore at 16-SEP-11
using target database control file instead of recovery catalog
allocated channel: ORA DISK 1
channel ORA DISK 1: sid=155 devtype=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: sid=154 devtype=DISK
allocated channel: ORA DISK 3
channel ORA DISK 3: sid=153 devtype=DISK

channel ORA_DISK_1: starting datafile backupset restore
channel ORA_DISK_1: specifying datafile(s) to restore from backup set
restoring datafile 00004 to /u01/app/oracle/oradata/TEST/users01.dbf
channel ORA DISK 1: reading from backup piece
/u01/app/oracle/flash recovery area/TEST/backupset/2011 09 16/o1 mf nnndf
TAG20110916T1133
10 775jvqnj .bkp
channel ORA DISK 1: restored backup piece 1
piecehandle=/u01/app/oracle/flash recovery area/TEST/backupset/2011 09 16/
o1_mf_nnndf_TAG20
110916T113310_775jvqnj_.bkp tag=TAG20110916T113310
channel ORA DISK 1: restore complete, elapsed time: 00:00:01
Finished restore at 16-SEP-11
```

此时，数据文件 users01.dbf 已经恢复到原来目录，我们通过操作系统指令来确认。

```
[oracle@ocm1 ~]$ ls /u01/app/oracle/oradata/TEST
control01.ctl  example01.dbf  redo03.log    temp01.dbf
control02.ctl  redo01.log     sysaux01.dbf  undotbs01.dbf
control03.ctl  redo02.log     system01.dbf  users01.dbf
```

接着，我们使用 RECOVER 指令恢复数据文件，这个过程是使用重做日志文件记录的过程，包括前滚和回滚两个步骤。

例子 22-60　使用 RECOVER 指令恢复数据文件

```
RMAN> recover datafile 4;
```

```
Starting recover at 16-SEP-11
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3

starting media recovery
media recovery complete, elapsed time: 00:00:02

Finished recover at 16-SEP-11
```

显然，这个恢复过程成功了，因为我们没有覆盖重做日志文件，自备份以来所有的数据库变化都记录在案，如果重做日志组被覆盖，则指令 recover datafile 4 可能会失败，此时我们只能使用如下指令实现不完全恢复。

```
SQL>RECOVER  DATAFILE 4 UNTIL CALCEL
```

根据提示，直接输入 cancel，在介质恢复结束后，数据库可以以 resetlogs 打开。

在恢复了数据文件 4 后，我们可以打开数据库了。

```
RMAN> alter database open;

database opened
```

22.15.3 联机重做日志文件和数据文件损坏的恢复

这个案例模拟的是控制文件没有丢失，但是所有的重做日志文件以及数据文件损坏，我们先分析一下，这种情况下很有可能出现数据丢失，但也有可能不会丢失数据，比如数据库 DML 操作很少或者在一个很短的时间内（从备份到文件损坏），就没有执行任何 DML 操作，数据库处于静止状态，此时虽然数据文件丢失，但是不影响完全恢复，因为自备份以来没有数据变化，当然这种情况很少（但存在）。

在这种情况下，我们需要复原数据文件，然后恢复数据文件，但是由于所有重做日志丢失，我们只能使用 CANCEL 参数，让 RMAN 自己确定何时停止恢复。然后使用 RESETLOGS 打开数据库，此时会重建所有的重做日志文件。我们模拟这个故障，删除掉所有重做日志文件以及数据文件。

```
[oracle@ocm1 ~]$ cd /u01/app/oracle/oradata/TEST
[oracle@ocm1 TEST]$ rm -rf *.dbf
[oracle@ocm1 TEST]$ rm -rf *.log
[oracle@ocm1 TEST]$ ls
control01.ctl  control02.ctl  control03.ctl
```

我们删除了所有的数据文件，以及重做日志文件。重新启动数据库看如何报错。

```
SQL> startup
ORACLE instance started.

Total System Global Area  2195212672 bytes
Fixed Size                 12119992 bytes
Variable Size             713047194 bytes
Database Buffers          209715200 bytes
```

```
    Redo Buffers                2973696 bytes
    Database mounted.
    ORA-01157: cannot identify/lock data file 1 - see DBWR trace file
    ORA-01110: data file 1: '/u01/app/oracle/oradata/TEST/system01.dbf
```

由于控制文件没有丢失，所以数据库知道数据文件的位置，当试图打开控制文件中记录的第一个数据文件时发现该文件不存在，数据库无法启动。如果是非当前的默认永久表空间，或者不是 SYSTEM 表空间，UNDO 表空间，则可以使用 alter database datafile …offline 的方式，先使得数据文件离线，然后打开数据库，这样数据库依然可以对外提供部分业务，不至于一个数据文件损坏造成整个数据库业务的中断。

下面我们使用 RMAN 复原（RESTORE）数据库。

例子 22-61　使用 RMAN 复原（RESTORE）数据库

```
    RMAN> restore database;

    Starting restore at 16-SEP-11
    using target database control file instead of recovery catalog
    allocated channel: ORA_DISK_1
    channel ORA_DISK_1: sid=155 devtype=DISK
    allocated channel: ORA_DISK_2
    channel ORA_DISK_2: sid=154 devtype=DISK
    allocated channel: ORA_DISK_3
    channel ORA_DISK_3: sid=153 devtype=DISK

    channel ORA_DISK_3: starting datafile backupset restore
    channel ORA_DISK_3: specifying datafile(s) to restore from backup set
    restoring datafile 00001 to /u01/app/oracle/oradata/TEST/system01.dbf
    channel ORA_DISK_3: reading from backup piece
    /u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/o1_mf_nnndf_
TAG20110916T1242
    37_775nxxg2_.bkp
    channel ORA_DISK_1: starting datafile backupset restore
    channel ORA_DISK_1: specifying datafile(s) to restore from backup set
    restoring datafile 00002 to /u01/app/oracle/oradata/TEST/undotbs01.dbf
    restoring datafile 00005 to /u01/app/oracle/oradata/TEST/example01.dbf
    channel ORA_DISK_1: reading from backup piece
    /u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/o1_mf_nnndf_
TAG20110916T1242
    37_775nxxr5_.bkp
    channel ORA_DISK_2: starting datafile backupset restore
    channel ORA_DISK_2: specifying datafile(s) to restore from backup set
    restoring datafile 00003 to /u01/app/oracle/oradata/TEST/sysaux01.dbf
    restoring datafile 00004 to /u01/app/oracle/oradata/TEST/users01.dbf
    channel ORA_DISK_2: reading from backup piece
    /u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/o1_mf_nnndf_
TAG20110916T1242
    37_775nxxnb_.bkp
    channel ORA_DISK_2: restored backup piece 1
    piecehandle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/
o1_mf_nnndf_TAG20
```

```
    110916T124237 775nxxnb .bkp tag=TAG20110916T124237
    channel ORA_DISK_2: restore complete, elapsed time: 00:00:46
    channel ORA_DISK_1: restored backup piece 1
    piecehandle=/u01/app/oracle/flash recovery area/TEST/backupset/2011 09 16/
o1 mf nnndf TAG20
    110916T124237_775nxxr5_.bkp tag=TAG20110916T124237
    channel ORA_DISK_1: restore complete, elapsed time: 00:00:53
    channel ORA DISK 3: restored backup piece 1
    piecehandle=/u01/app/oracle/flash recovery area/TEST/backupset/2011 09 16/
o1 mf nnndf TAG20
    110916T124237_775nxxg2_.bkp tag=TAG20110916T124237
    channel ORA_DISK_3: restore complete, elapsed time: 00:01:119
    Finished restore at 16-SEP-11
```

在成功复原数据文件后，下面使用 RECOVER 指令恢复数据库。

例子 22-62　使用 RECOVER 指令恢复数据库

```
    SQL> recover database until cancel;
    ORA-00279: change 573315 generated at 09/16/2011 13:119:29 needed for thread
1
    ORA-002199: suggestion :
    /u01/app/oracle/flash_recovery_area/TEST/archivelog/2011_09_16/o1_mf_1_1_%
u .arcORA-002190: change 573315 for thread 1 is in sequence #1


    Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
    cancel
    Media recovery cancelled. 1
```

此时，重做日志文件依然没有恢复，在打开数据库时使用 RESETLOGS 指令创建依据控制文件中记录的重做日志组信息重建日志文件。先看如果不使用 RESETLOGS 会提示什么。

```
    SQL> alter database open;
    alter database open
    *
    ERROR at line 1:
    ORA-00313: open failed for members of log group 1 of thread 1
    ORA-00312: online log 1 thread 1: '/u01/app/oracle/oradata/TEST/redo01.log'
```

此时，提示无法打开日志文件，显然虽然控制文件中记录了重做日志信息，但是实际上该文件已经不在控制文件中记录的位置，所以在试图打开第一个日志组日志文件时就报错。下面我们继续使用 RESETLOGS 打开数据库，此时会创建新的重做日志组。

```
    SQL> alter database open resetlogs;

    Database altered.
```

这种情况下，丢失数据是难免的，因为自备份以来所有的数据变化都丢失了，这是很严重的情况，所以在数据库管理中要格外注意重做日志要有多组，每组要设置不同路径的多个日志文件，实现重做日志的冗余设置，提高数据库系统的可靠性。

22.15.4 如何将数据文件恢复到其他磁盘目录下

在数据文件损坏后，如果是磁盘损坏此时无法再使用，如果更新磁盘显然时间上不现实，很多数据库系统无法容忍过长的停机时间。此时，可以将数据文件恢复到指定目录。比如 SYSTEM 数据文件和 USERS 表空间的数据文件损坏，而且当前磁盘无法使用，我们在将数据库启动到 MOUNT 状态后，使用如下指令恢复。

例子 22-63 使用如下指令恢复数据文件

```
    RMAN> run {
    2> set newname for datafile
    3>'/u01/app/oracle/oradata/TEST/system01.dbf' to '/u02/app/oracle/oradata/
TEST/system01.dbf;
    4> set newname for datafile
    5>'/u01/app/oracle/oradata/TEST/users01.dbf' to '/u02/app/oracle/oradata/
TEST/users01.dbf';
    6> restore database from tag=TAG20110916T131914;
    7> switch datafile all;
    8> }
```

然后使用 recover database until cancel 来恢复数据库，使用 alter database open resetlogs 来打开数据库。

22.16 使用RMAN归档模式下的完全恢复

在归档模式下，使用 RMAN 的备份和所有归档重做日志以及当前的重做日志文件可以实现数据库的完全恢复。要求在使用 RMAN 备份以来数据库一直运行在归档模式，且归档文件以及重做日志文件没有损坏。这种情况下可以联机恢复数据库文件，不需要关闭数据库。在生产库中，联机恢复的最大好处就是不影响其他业务。下面是几种情形，演示数据库相关文件损坏后如何在归档模式下实现完全恢复。

22.16.1 非系统表空间损坏的恢复

这个案例的背景是数据库运行在归档模式，表空间 USERS 的数据文件损坏，有完整的备份，以及当前归档的日志文件完好。这种情况下，为了保证数据库对外业务的不中断，首先将该表空间 OFFLINE，然后使用备份复原数据文件，再使用归档日志、当前的重做日志恢复数据文件。下面演示这个过程。

首先删除数据表空间 USERS 的数据文件。

```
    [oracle@ocm1 TEST]$ ls
    control01.ctl   control03.ctl   redo01.log   redo03.log      system01.dbf
undotbs01.dbf
    control02.ctl    example01.dbf    redo02.log    sysaux01.dbf    temp01.dbf
users01.dbf
    [oracle@ocm1 TEST]$ rm -rf users01.dbf
```

重新启动数据库，会报错，如下所示。

```
SQL> startup
ORACLE instance started.

Total System Global Area  2195212672 bytes
Fixed Size                  12119992 bytes
Variable Size              713047194 bytes
Database Buffers           209715200 bytes
Redo Buffers                 2973696 bytes
Database mounted.
ORA-01157: cannot identify/lock data file 4 - see DBWR trace file
ORA-01110: data file 4: '/u01/app/oracle/oradata/TEST/users01.dbf'
```

为了先打开数据库，先将该数据文件离线，然后打开数据库。

例子 22-64　将该数据文件离线后打开数据库

```
SQL> alter database datafile 4 offline;

Database altered.

SQL> alter database open;

Database altered.
```

启动 RMAN 复原数据文件 4。

例子 22-65　启动 RMAN 复原数据文件 4

```
RMAN> restore datafile 4;

Starting restore at 16-SEP-11
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=143 devtype=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: sid=142 devtype=DISK
allocated channel: ORA_DISK_3
channel ORA_DISK_3: sid=141 devtype=DISK

channel ORA_DISK_1: starting datafile backupset restore
channel ORA_DISK_1: specifying datafile(s) to restore from backup set
restoring datafile 00004 to /u01/app/oracle/oradata/TEST/users01.dbf
channel ORA_DISK_1: reading from backup piece
/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/o1_mf_nnndf_TAG2
0110916T153424_775z00on_.bkp
channel ORA_DISK_1: restored backup piece 1
piece
handle=/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/o1_mf_nnn
df_TAG20110916T153424_775z00on_.bkp tag=TAG20110916T153424
channel ORA_DISK_1: restore complete, elapsed time: 00:00:02
Finished restore at 16-SEP-11
```

恢复数据文件，此时会应用归档日志或者重做日志数据恢复数据文件 4。

例子 22-66　应用归档日志或者重做日志数据恢复数据文件 4

```
RMAN> recover datafile 4;

Starting recover at 16-SEP-11
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3

starting media recovery
media recovery complete, elapsed time: 00:00:03

Finished recover at 16-SEP-11
```

此时，数据文件 4 完全恢复了，我们将数据文件 ONLINE，然后查看表 t21 是否恢复过来。

例子 22-67　将数据文件 ONLINE

```
SQL> alter database datafile 4 online;

Database altered.

SQL>   select   file_name,tablespace_name,status   ,online_status   from
dba_data_files;

FILE_NAME                        TABLESPACE_NAME     STATUS    ONLINE
-------------------------------- -------------------- --------- -------
/u01/app/oracle/oradata/TEST/users01.dbf   USERS      AVAILABLE ONLINE
/u01/app/oracle/oradata/TEST/sysaux01.dbf  SYSAUX     AVAILABLE ONLINE
/u01/app/oracle/oradata/TEST/undotbs01.dbf UNDOTBS1   AVAILABLE ONLINE
/u01/app/oracle/oradata/TEST/system01.dbf  SYSTEM     AVAILABLE SYSTEM
/u01/app/oracle/oradata/TEST/example01.dbf EXAMPLE    AVAILABLE ONLINE
```

再将数据文件 ONLINE 之后，数据文件可以被读写，通过下面查询知道表 t21 已经恢复。

例子 22-68　查询表 t21 验证恢复

```
SQL> select count(*) from t21;

  COUNT(*)
----------
     45619
```

复原并恢复数据文件时，也可以在 RMAN 中使用如下方式完成整个过程，通过运行 run{}块完成，run 内的所有操作都作为一个事务出现，如例子 22-69 所示。

例子 22-69　通过运行 run{}块复原并恢复数据文件

```
RMAN> run {
2> sql 'alter database datafile 4 offline';
3> restore tablespace users;
4> recover tablespace users;
5>  sql 'alter database datafile 4 online';
```

```
6> }
```

22.16.2　系统表空间损坏的恢复

SYSTEM 表空间损坏，而控制文件、重做日志文件完好，此时需要把数据库启动到 MOUNT 状态，使用 RMAN 恢复该表空间。恢复过程与恢复非系统表空间类似，区别是必须在数据库 MOUNT 时恢复，因为 SYSTEM 表空间损坏数据库根本无法启动。

SYSTEM 表空间数据文件丢失，数据库立即关闭，重启数据库提示如下错误。

```
SQL> startup
ORACLE instance started.

Total System Global Area  2195212672 bytes
Fixed Size                  12119992 bytes
Variable Size              713047194 bytes
Database Buffers           209715200 bytes
Redo Buffers                 2973696 bytes
Database mounted.
ORA-01157: cannot identify/lock data file 1 - see DBWR trace file
ORA-01110: data file 1: '/u01/app/oracle/oradata/TEST/system01.dbf'
```

在数据库处于 MOUNT 状态时，使用 RMAN 恢复 SYSTEM 表空间，如下所示。

例子 22-70　使用 RMAN 恢复 SYSTEM 表空间

```
RMAN> run {
2> sql 'alter database datafile 1 offline';
3> restore datafile 1;
4> recover datafile 1;
5> sql 'alter database datafile 1 online';
6> }

using target database control file instead of recovery catalog
sql statement: alter database datafile 1 offline

Starting restore at 16-SEP-11
allocated channel: ORA DISK 1
channel ORA_DISK_1: sid=155 devtype=DISK
allocated channel: ORA DISK 2
channel ORA DISK 2: sid=154 devtype=DISK
allocated channel: ORA DISK 3
channel ORA_DISK_3: sid=153 devtype=DISK

channel ORA DISK 1: starting datafile backupset restore
channel ORA DISK 1: specifying datafile(s) to restore from backup set
restoring datafile 00001 to /u01/app/oracle/oradata/TEST/system01.dbf
channel ORA_DISK_1: reading from backup piece
/u01/app/oracle/flash_recovery_area/TEST/backupset/2011_09_16/o1_mf_nnndf_
TAG20110916T1534
24 775z00n9 .bkp
channel ORA_DISK_1: restored backup piece 1
```

```
    piecehandle=/u01/app/oracle/flash recovery area/TEST/backupset/2011 09 16/
o1_mf_nnndf_TAG20
    110916T153424_775z00n9_.bkp tag=TAG20110916T153424
    channel ORA DISK 1: restore complete, elapsed time: 00:00:35
    Finished restore at 16-SEP-11

    Starting recover at 16-SEP-11
    using channel ORA DISK 1
    using channel ORA DISK 2
    using channel ORA DISK 3

    starting media recovery
    media recovery complete, elapsed time: 00:00:01

    Finished recover at 16-SEP-11

    sql statement: alter database datafile 1 online
```

在 RMAN 复原并恢复数据文件 1 后，就可以打开数据库了，如下所示。

例子 22-71　打开数据库

```
SQL> alter database open;

Database altered.
```

此时，完全恢复 SYSTEM 表空间。

22.16.3　所有数据文件丢失的恢复

在这个案例中，所有数据文件丢失，但是控制文件与重做日志文件都完好，这种情况使用 RMAN 恢复很方便。请读者注意，归档模式下的完全恢复的前提是有备份以及日志文件完好。所以本节给出的几个例子背景相似，只是丢失的文件类型不同。

例子 22-72　所有数据文件丢失的例子

```
RMAN> run {
2> restore database;
3> recover database;
4> sql 'alter database open';
5> }
```

22.17　RMAN实现数据块恢复

使用 RMAN 可以实现数据块级的数据恢复，在传统恢复手段中即某个数据文件的一个数据块被损坏，就造成整个数据文件无法使用，此时必须通过备份恢复整个数据文件，显然这样的方法恢复时间较长，而 RMAN 实现块级恢复，如果某个数据文件的数据块损坏，通过数据文件的完整备份就可以恢复数据块。下面我们演示如何使用 RMAN 恢复数据块。

首先需要打开 RMAN 然后备份整个数据库，这是一个全备份。

例子 22-73　打开 RMAN

```
D:\>rman target /

恢复管理器: Release 11.2.0.1.0 - Production on 星期日 7月 11 16:04:57 2010

Copyright (c) 1992, 2005, Oracle.  All rights reserved.

连接到目标数据库: ORCL (DBID=12511947545)
```

例子 22-74　备份整个数据库

```
RMAN> backup database plus archivelog;


启动 backup 于 11-7月 -10
当前日志已存档
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: sid=136 devtype=DISK
通道 ORA_DISK_1: 正在启动存档日志备份集
通道 ORA_DISK_1: 正在指定备份集中的存档日志
输入存档日志线程 =1 序列 =4 记录 ID=1 时间戳=724019956
通道 ORA_DISK_1: 正在启动段 1 于 11-7月 -10
通道 ORA_DISK_1: 已完成段 1 于 11-7月 -10
段句柄
=E:\ORACLE\PRODUCT\11.2.0\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\2010_07_11\O
1_MF_ANNNN_TAG20100711T160557_63LYV6NJ_.BKP 标 记 =TAG20100711T160557 注 释
=NONE
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:02
完成 backup 于 11-7月 -10

启动 backup 于 11-7月 -10
………..
………..
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:02
完成 backup 于 11-7月 -10
```

此时，我们将归档日志也放在备份集中，这不是必须的，读者可以自行选择，备份文件的存储目录我们使用 Oracle 默认的快闪恢复区。

接着就是做些破坏性行为，我们先关闭数据库。

例子 22-75　关闭数据库并使用 ULTRAEDIT 修改数据文件 RMAN.DBF

```
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
```

数据库关闭后，使用 ULTRAEDIT 编辑数据文件 RMAN.DBF，破坏一些数据，并保存来模拟数据文件的损坏。如果我们再启动数据库就会报错，如例子 22-75 所示。

例子 22-76　数据文件 6 损坏后启动数据库

```
SQL> startup
```

```
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                 12503190 bytes
Variable Size             176163764 bytes
Database Buffers          419430400 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
ORA-01113: 文件 6 需要介质恢复
ORA-01110: 数据文件 6: 'D:\RMAN.DBF'
```

进行数据文件 6 的有效性检验，如果该数据文件中有坏块，则在数据字典视图 v$database_block_corruption 中有详细记录。

例子 22-77　对数据文件 6 进行有效检验

```
RMAN> backup validate datafile 6;

启动 backup 于 11-7 月 -10
使用目标数据库控制文件替代恢复目录
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: sid=155 devtype=DISK
通道 ORA_DISK_1: 启动全部数据文件备份集
通道 ORA_DISK_1: 正在指定备份集中的数据文件
输入数据文件 fno=00006 name=D:\RMAN.DBF
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:02
完成 backup 于 11-7 月 -10
```

然后，我们通过数据字典 v$database_block_corruption 来查看数据文件 6 中损坏的数据块，如例子 22-78 所示。

例子 22-78　查看数据文件 6 中损坏的数据块

```
SQL> select *
  2  from v$database_block_corruption;

     FILE#     BLOCK#     BLOCKS CORRUPTION CHANGE# CORRUPTIO
---------- ---------- ---------- ------------------ ---------
         6        118          1                  0 CHECKSUM
```

从输出看出数据文件 6 的数据块 118 被损坏，所以需要修复。也可以查看告警日志文件，日志文件中记录了数据文件 6 的坏块信息，如下所示。

```
Hex dump of (file 6, block 118) in trace file
e:\oracle\product\10.2.0\admin\orcl\udump\orcl ora 2872.trc
Corrupt block relative dba: 0x011900076 (file 6, block 118)
Bad check value found during backing up datafile
Data in bad block:
 type: 0 format: 2 rdba: 0x00000076
 last change scn: 0x0000.00000000 seq: 0x1 flg: 0x05
 spare1: 0x0 spare2: 0x0 spare3: 0x0
 consistency value in tail: 0x00000001
 check value in block header: 0xa75f
```

```
 computed block checksum: 0x29
Reread of blocknum=118, file=D:\RMAN.DBF. found same corrupt data
Reread of blocknum=118, file=D:\RMAN.DBF. found same corrupt data
Reread of blocknum=118, file=D:\RMAN.DBF. found same corrupt data
Reread of blocknum=118, file=D:\RMAN.DBF. found same corrupt data
Reread of blocknum=118, file=D:\RMAN.DBF. found same corrupt data
```

无论通过数据字典视图还是告警日志文件，我们都可以确定数据文件 6 的 118 号数据块损坏。我们通过 RMAN 恢复这个数据块，如例子 22-79 所示。

例子 22-79　使用 RMAN 完成数据块恢复

```
RMAN> blockrecover datafile 6 block 118 from backupset;

启动 blockrecover 于 11-7 月 -10
使用通道 ORA_DISK_1

通道 ORA DISK 1: 正在恢复块
通道 ORA DISK 1: 正在指定要从备份集恢复的块
正在恢复数据文件 00006 的块
通道 ORA_DISK_1: 正在读取备份段
E:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\2010_07_11\O1_
MF NNNDF TAG20
100711T160559 63LYV19J3 .BKP
通道 ORA_DISK_1: 已从备份段 1 恢复块
段句柄 =
E:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\BACKUPSET\2010 07 11\O1
MF NNNDF TAG20100711T160559 63LYV19J3
.BKP 标记 = TAG20100711T160559
通道 ORA_DISK_1: 块恢复完成, 用时: 00:00:02

正在开始介质的恢复
某些块未恢复: 有关详细信息, 请参阅跟踪文件
介质恢复完成, 用时: 00:00:01

完成 blockrecover 于 11-7 月 -10
```

此时，提示已经成功恢复数据块。但是由于某些块未恢复所以数据文件验证失败，可以通过数据告警日志看到这个提示。我们查看告警日志文件了解这个恢复过程。

```
Starting block media recovery
Sun Jul 11 16:40:27 2010
Recovery of Online Redo Log: Thread 1 Group 1 Seq 5 Reading mem 0
  Mem# 0 errs 0: E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO01.LOG
Sun Jul 11 16:40:27 2010
Recovery of Online Redo Log: Thread 1 Group 2 Seq 6 Reading mem 0
  Mem# 0 errs 0: E:\ORACLE\PRODUCT\11.2.0\ORADATA\ORCL\REDO02.LOG
Sun Jul 11 16:40:27 2010
Errors in file e:\oracle\product\11.2.0\admin\orcl\udump\orcl ora 2872.trc:
ORA-01122: 数据库文件 6 验证失败
ORA-01110: 数据文件 6: 'D:\RMAN.DBF'
ORA-01208: 数据文件是旧的版本 - 不能访问当前版本
```

```
Sun Jul 11 16:40:27 2010
Completed block media recovery
```

此时提示数据文件验证失败，所以在打开数据库前必须 RECOVER 数据文件 6，如例子 22-80 所示。

例子 22-80　恢复数据文件 6

```
SQL> recover datafile 6;
完成介质恢复。
```

然后，打开数据库并验证表 RMAN_EMP 是否存在。

例子 22-81　打开数据库并验证表 RMAN_EMP 是否恢复

```
SQL> alter database open;

数据库已更改。
SQL> select count(ename)
  2  from rman_emp;

COUNT(ENAME)
------------
          14
```

此时，我们成功打开了数据库并且通过查询验证了数据文件 6 中唯一的表已经恢复，而且没有任何数据丢失。

22.18　RMAN的备份维护指令

本节我们介绍几个常用的 RMAN 验证指令，分别是 VALIDATE BACKUPSET、RESTORE…VALIDATE、RESTORE…PREVIEW、LIST、REPORT。下面分别详细介绍这些指令的作用以及用法。

22.18.1　RMAN 的 VALIDATE BACKUPSET 指令

在使用 RMAN 备份了数据库后，需要恢复时最好使用 VALIDATE BACKUPSET 指令验证备份文件的可用性，如备份的数据文件都以备份集的形式存在，在使用 VALIDATE BACKUPSET 验证备份集时 RMAN 会自动找到你指定的备份集，如例子 22-82 所示。

例子 22-82　使用 VALIDATE BACKUPSET 指令验证备份集的可用性

```
RMAN> validate backupset 5;

using channel ORA DISK 1
channel ORA_DISK_1: starting validation of archive log backupset
channel ORA_DISK_1: reading from backup piece
/u01/app/oracle/oracle/product/11.2.0/db 1/dbs/05mdle4g 1 1
channel ORA DISK 1: restored backup piece 1
piece handle=/u01/app/oracle/oracle/product/11.2.0/db_1/dbs/05mdle4g_1_1
```

```
tag=TAG20110530T203320
channel ORA_DISK_1: validation complete, elapsed time: 00:00:01
```

在例子 22-82 中，RMAN 确实启动了数据文件备份集验证，而且“验证完成”说明成功此备份集是有效地，可用于恢复操作。读者应该会问数字 5 代表什么，其实它代表了 RMAN 的所有备份集中代表某个备份集的关键字。

使用如下 LIST BACKUP SUMMARY 指令查看备份集的汇总信息。

例子 22-83　查看备份集汇总信息

```
RMAN> list backup summary;


备份列表
===============
关键字  TY LV  S 设备类型      完成时间        段数 副本数  压缩标记
---------- --- ---------------------- -------- --------------------------
1       B  F  A DISK        22-9月 -09 1     1   NO      TAG20090822T21595
..........
30      B  F  A DISK        01-9月 -09 1     1   NO      TAG20090901T23193
31      B  F  A DISK        01-9月 -09 1     1   NO     TAG20090901T239419
```



使用 VALIDATE BACKUPSET 只是验证了备份集中某个备份集的有效性，但是如果要知道某个表空间或数据文件是否在备份集中又该如何处理呢，Oracle 提供了 RESTORE...VALIDATE 指令。

22.18.2　RMAN 的 RESTORE...VALIDATE 指令

RMAN 支持使用 RESTORE...VALIDATE 验证数据库对象是否在当前的备份集中，这样在用户恢复一个数据文件或一个表空间时，可以首先确认该对象备份信息是否存在。

例子 22-84　验证表空间 SYSAUX 备份信息是否在备份集中

```
RMAN> restore tablespace users validate;

Starting restore at 29-SEP-11
using channel ORA DISK 1

channel ORA_DISK_1: starting validation of datafile backupset
channel ORA DISK 1: reading from backup piece
/u01/app/oracle/oracle/product/11.2.0/db 1/dbs/0bmnpitm 1 1
channel ORA_DISK_1: restored backup piece 1
piece handle=/u01/app/oracle/oracle/product/11.2.0/db_1/dbs/0bmnpitm_1_1
tag=TAG20110929T190213
channel ORA DISK 1: validation complete, elapsed time: 00:01:07
Finished restore at 29-SEP-11
```

下面再给出验证一个数据文件是否在备份集中的例子。

例子 22-85　验证数据文件是否在备份集中

```
RMAN> restore datafile '/u01/app/oracle/oradata/PROD/system01.dbf' validate;

Starting restore at 29-SEP-11
using channel ORA DISK 1

channel ORA_DISK_1: starting validation of datafile backupset
channel ORA_DISK_1: reading from backup piece
/u01/app/oracle/oracle/product/10.2.0/db 1/dbs/0bmnpitm 1 1
channel ORA DISK 1: restored backup piece 1
piece handle=/u01/app/oracle/oracle/product/10.2.0/db 1/dbs/0bmnpitm 1 1
 tag=TAG20110929T190213
channel ORA_DISK_1: validation complete, elapsed time: 00:01:07
Finished restore at 29-SEP-11
```

上述输出中的“验证完成”说明该数据文件存在于备份集中，当然也可以使用数据文件标识如 restore datafile 1 validate，验证数据文件 1 是否在备份集中，是否可恢复。

22.18.3　RMAN 的 RESTORE…PREVIEW 指令

用户在备份数据库前，或许想知道执行恢复的所有文件是否存在，如当恢复表空间时，想知道该表空间中的所有数据文件是否在备份集中，在恢复全库时刻的数据文件归档日志文件是否存在等等。RMAN 提供了 RESTORE…PREVIEW 指令完成这项功能。

例子 22-86　查看恢复整个数据库所需的备份文件是否存在

```
RMAN> restore database preview;

Starting restore at 29-SEP-11
using channel ORA DISK 1


List of Backup Sets
===================

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
11      Full    152.16M    DISK        00:03:44     29-SEP-11
        BP Key: 11   Status: AVAILABLE  Compressed: YES  Tag: TAG20110929T190213
Piece Name: /u01/app/oracle/oracle/product/11.2.0/db_1/dbs/0bmnpitm_1_1
  List of Datafiles in backup set 11
  File LV Type Ckp SCN    Ckp Time  Name
  ---- -- ---- ---------- --------- ----
  1       Full 1184936    29-SEP-11 /u01/app/oracle/oradata/PROD/system01.dbf
  2       Full 11194936   29-SEP-11 /u01/app/oracle/oradata/PROD/undotbs01.dbf
  3       Full 1184936    29-SEP-11 /u01/app/oracle/oradata/PROD/sysaux01.dbf
  4       Full 1184936    29-SEP-11 /u01/app/oracle/oradata/PROD/users01.dbf
  7       Full 1184936    29-SEP-11 /u01/app/oracle/oradata/PROD/users02.dbf
  19      Full 1184936    29-SEP-11 /u01/app/oracle/oradata/PROD/audit01.dbf
```

```
List of Archived Log Copies
Key     Thrd Seq     S Low Time  Name
------- ---- ------- - --------- ----
55      1    69      A 29-SEP-11
/u01/app/oracle/oracle/product/11.2.0/db_1/dbs/arch1_69_7439797196.dbf
56      1    70      A 29-SEP-11
/u01/app/oracle/oracle/product/11.2.0/db_1/dbs/arch1_70_7439797196.dbf
Media recovery start SCN is 1184936
Recovery must be done beyond SCN 1184936 to clear data files fuzziness
Finished restore at 29-SEP-11
```

最后显示的“完成 restore”说明数据库所需要的备份文件都存在。同样可以验证恢复某个表空间或数据文件所需的备份文件是否存在，如下所示。

```
RMAN> restore tablespace sysaux preview;
RMAN> restore datafile 5 preview;
```

上述输出中的“验证完成”说明该数据文件存在于备份集中。

22.18.4　RMAN 的 LIST 指令

RMAN 的 LIST 指令可以查看当前的备份集信息，某个表空间所在的备份集，某个数据文件所在的备份集等信息，在恢复之前使用 LIST 指令可以清楚地了解所需要的备份是否存在，以及存在几个备份，不同的备份集具有不同的备份集标识。我们通过几个例子来演示。

首先看 LIST 指令下具有哪几个指令。

```
RMAN> list ;

RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-00558: error encountered while parsing input commands
RMAN-01009: syntax error: found ";": expecting one of: "all, archivelog, backup,
backuppiece, backupset, backed, completed, copy, controlfilecopy, datafilecopy,
device, expired, global, incarnation, like, proxy, recoverable, script, tag"
RMAN-01007: at line 1 column 6 file: standard input
```

这虽然是一个错误提示，但是在告诉我们语法错误的同时，使用这个提示也就知道 list 之后需要哪些指令，这样可以逐层查询需要的指令。下面通过几个例子演示。

查看备份集信息。

例子 22-87　查看备份集信息

```
RMAN> list backupset;


List of Backup Sets
===================

BS Key  Size       Device Type Elapsed Time Completion Time
------- ---------- ----------- ------------ ---------------
```

```
    5       1945.50K    DISK        00:00:01     30-MAY-11
            BP Key: 5   Status: AVAILABLE  Compressed: YES  Tag: TAG20110530T203320
            Piece Name: /u01/app/oracle/oracle/product/11.2.0/db_1/dbs/05mdle4g_1_1

      List of Archived Logs in backup set 5
      Thrd Seq     Low SCN    Low Time  Next SCN   Next Time
      ---- ------- ---------- --------- ---------- ---------
      1    53      913472     30-MAY-11 914149     30-MAY-11

    BS Key  Type LV Size       Device Type Elapsed Time Completion Time
    ------- ---- -- ---------- ----------- ------------ ---------------
    9       Full    152.09M    DISK        00:03:17     29-SEP-11
            BP Key: 9   Status: AVAILABLE  Compressed: YES  Tag: TAG20110929T1051945
            Piece                                                                Name:
/u01/app/oracle/oracle/product/11.2.0/db 1/dbs/09mnomj5 1 1
      List of Datafiles in backup set 9
      File LV Type Ckp SCN    Ckp Time  Name
      ---- -- ---- ---------- --------- ----
      1       Full 11622919   29-SEP-11 /u01/app/oracle/oradata/PROD/system01.dbf
      2       Full 11622919   29-SEP-11 /u01/app/oracle/oradata/PROD/undotbs01.dbf
      3       Full 11622919   29-SEP-11 /u01/app/oracle/oradata/PROD/sysaux01.dbf
      4       Full 11622919   29-SEP-11 /u01/app/oracle/oradata/PROD/users01.dbf
      7       Full 11622919   29-SEP-11 /u01/app/oracle/oradata/PROD/users02.dbf
      19      Full 11622919   29-SEP-11 /u01/app/oracle/oradata/PROD/audit01.dbf

    BS Key  Type LV Size       Device Type Elapsed Time Completion Time
    ------- ---- -- ---------- ----------- ------------ ---------------
    10      Full    7.95M      DISK        00:00:02     29-SEP-11
            BP Key: 10   Status: AVAILABLE  Compressed: NO  Tag: TAG20110929T110211
            Piece                                                                Name:
/u01/app/oracle/oracle/product/11.2.0/db 1/dbs/c-15519119122-2 0110929-00
      Control File Included: Ckp SCN: 1162372      Ckp time: 29-SEP-11
      SPFILE Included: Modification time: 29-SEP-11
```

输出显示了详细的备份集信息，当前的 RMAN 备份包含两个备份集，一个是 BS 5，一个是 BS 9，我们也可以通过备份集标识来查看备份集，如例子 22-88 所示。

例子 22-88　通过备份集标识来查看备份集

```
    RMAN> list backupset 5;


    List of Backup Sets
    ===================

    BS Key  Size       Device Type Elapsed Time Completion Time
    ------- ---------- ----------- ------------ ---------------
    5       1945.50K    DISK        00:00:01     30-MAY-11
            BP Key: 5   Status: AVAILABLE  Compressed: YES  Tag: TAG20110530T203320
            Piece Name: /u01/app/oracle/oracle/product/11.2.0/db 1/dbs/05mdle4g 1 1

      List of Archived Logs in backup set 5
      Thrd Seq     Low SCN    Low Time  Next SCN   Next Time
```

```
  ---- ------- ---------- --------- ---------- ---------
  1    53      913472     30-MAY-11 914149     30-MAY-11
```

只查看备份集 5 的信息，从输出知道该备份集是归档日志文件的备份集。列出某个表空间在备份集中的信息。

例子 22-89 使用 List 列出备份集

```
RMAN> list backup of tablespace users;


List of Backup Sets
===================

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
9       Full    152.09M    DISK        00:03:17     29-SEP-11
        BP Key: 9   Status: AVAILABLE  Compressed: YES  Tag: TAG20110929T1051945
        Piece                                                          Name:
/u01/app/oracle/oracle/product/11.2.0/db 1/dbs/09mnomj5 1 1
  List of Datafiles in backup set 9
  File LV Type Ckp SCN    Ckp Time  Name
  ---- -- ---- ---------- --------- ----
  4       Full 11622919   29-SEP-11 /u01/app/oracle/oradata/PROD/users01.dbf
  7       Full 11622919   29-SEP-11 /u01/app/oracle/oradata/PROD/users02.dbf
```

从输出知道表空间 USERS 的备份存放在备份集 9 中，该表空间的备份包含两个数据文件。查询某个数据文件在备份集中的信息。

例子 22-90 查询某个数据文件在备份集中的信息

```
RMAN> list backup of datafile 1;


List of Backup Sets
===================

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
9       Full    152.09M    DISK        00:03:17     29-SEP-11
        BP Key: 9   Status: AVAILABLE  Compressed: YES  Tag: TAG20110929T1051945
        Piece Name: /u01/app/oracle/oracle/product/11.2.0/db 1/dbs/09mnomj5 1 1
  List of Datafiles in backup set 9
  File LV Type Ckp SCN    Ckp Time  Name
  ---- -- ---- ---------- --------- ----
  1       Full 11622919   29-SEP-11 /u01/app/oracle/oradata/PROD/system01.dbf
```

使用 RMAN 的 LIST 指令，还可以查看诸如归档日志文件以及控制文件、参数文件的备份信息。

显示归档日志文件的备份信息。

```
RMAN>list backup of archivelog all;
RMAN>list backup of archivelog from time ='sysdate-2';
```

显示控制文件以及参数文件备份。

```
RMAN>list backup of controlfile;
RMAN>list backup of spfile;
RMAN>list copy of controlfile;
```

我们还可以查看备份的汇总信息，如下所示。

例子 22-91 查看备份的汇总信息

```
    RMAN> list backup summary;


    List of Backups
    ===============
    Key   TY LV S Device Type Completion Time #Pieces #Copies Compressed Tag
    ------- -- -- - ----------- --------------- ------- ------- ---------- ---
    5          B   A   A  DISK           30-MAY-11         1         1          YES
TAG20110530T203320
    9          B   F   A  DISK           29-SEP-11          1         1          YES
TAG20110929T1051945
    10         B   F   A  DISK           29-SEP-11          1         1           NO
TAG20110929T110211
```

其中 KEY 是备份集标识，B 表示备份集，LV 的 A 说明是归档文件，F 表示数据文件的完全备份，S 的 A 表示可用。

22.18.5 RMAN 的 REPORT 指令

List 命令只是简单地将 rman 的元数据检索并显示出来，rman 同时还提供了 report 命令，该命令具有一定的分析功能。

首先，显示数据库的结构。

例子 22-92 显示数据库的结构

```
RMAN> report schema;

Report of database schema

List of Permanent Datafiles
===========================
File Size(MB) Tablespace           RB  segs    Datafile Name
---- -------- -------------------- ------- ------------------------
1    1279     SYSTEM      ***       /u01/app/oracle/oradata/PROD/system01.dbf
2    216      UNDOTBS     ***       /u01/app/oracle/oradata/PROD/undotbs01.dbf
3    325      SYSAUX      ***       /u01/app/oracle/oradata/PROD/sysaux01.dbf
4    150      USERS       ***       /u01/app/oracle/oradata/PROD/users01.dbf
7    150      USERS       ***       /u01/app/oracle/oradata/PROD/users02.dbf
19   100      AUDIT_TBS   ***       /u01/app/oracle/oradata/PROD/audit01.dbf

List of Temporary Files
=======================
```

```
File Size(MB) Tablespace          Maxsize(MB) Tempfile Name
---- -------- -------------------- ----------- --------------------
1    75       TEMP          32767    /u01/app/oracle/oradata/PROD/temp01.dbf
2    100      TEMP          32767    /u01/app/oracle/oradata/PROD/temp02.dbf
```

上述输出列出了永久数据文件和临时文件。

下面列出需要备份的数据文件，该指令会与设置的 RMAN 备份策略相关联。

例子 22-93　列出需要备份的数据文件

```
RMAN> report need backup;

File #bkps Name
---- ----- -----------------------------------------------------
1    0     /u01/app/oracle/oradata/PROD/system01.dbf
2    0     /u01/app/oracle/oradata/PROD/undotbs01.dbf
3    0     /u01/app/oracle/oradata/PROD/sysaux01.dbf
4    0     /u01/app/oracle/oradata/PROD/users01.dbf
7    0     /u01/app/oracle/oradata/PROD/users02.dbf
8    0     /u01/app/oracle/oradata/PROD/audit01.dbf
```

说明与当前备份策略对比，这些数据文件需要备份，其实，在执行本指令前笔者将备份集全部删除（使用 RMAN 的 DELETE 指令），当前的保留策略是保留一份冗余，所以当发出上述指令时就显示这些数据文件需要备份。

22.19　本章小结

本章介绍了 RMAN 备份与恢复，RMAN 实现了数据库备份与恢复的自动化处理，并且支持增量备份，指令简洁，维护方便。重点是理解 RMAN 备份与恢复技术，掌握使用 RMAN 实现联机备份和脱机备份以及相应的恢复方法。理解归档模式和非归档模式下，数据库恢复的本质区别。并且 RMAN 提供了一系列的备份验证工具，使得在恢复前及明了解备份文件的状态。

第 23 章

Oracle闪回技术

Oracle 的闪回技术是一种数据恢复技术，具有恢复时间快，不使用备份文件的特点，它使得数据库可以回到过去的某个状态，可以满足用户的逻辑错误的快速恢复，需要注意闪回技术仅仅对逻辑恢复有效，如果是数据文件损坏必须使用介质恢复。本章我们讲解闪回技术的各种特性，或者说闪回的具体应用，一个是闪回版本查询、闪回事务查询，一个是闪回删除，最后一个是闪回数据库。

在没有闪回技术之前，数据库的逻辑错误恢复都是采用基于时间点的恢复，通过备份恢复数据库到过去指定的时间点，这种恢复方式需要使用备份并使用适当的归档日志完成，恢复时间取决于备份文件的复制时间和日志的应用时间，如果是很大的数据库可能会恢复很长时间，对于众多数据库应用系统来说，这是无法忍受的。而采用闪回技术，则可以更快速便捷地恢复用户错误或数据库逻辑错误。

23.1 理解闪回级别

闪回级别也可以理解为闪回粒度，针对闪回删除以及闪回数据库可以定义两种数据库的闪回级别，即表级闪回以及数据库级闪回。

- 数据库级闪回：数据库级闪回允许将整个数据库恢复到过去的某个时间点。数据库级的恢复在以下几种情况下使用。当误删除一个用户，或者误截断一个表时可以采用数据库级的闪回恢复。
- 表级闪回：表级闪回可以将表闪回到过去的某个时间点，或恢复到过去的某个 SCN，而闪回删除通过 DROP 指令删除的表。

下面将分别详细介绍闪回删除和闪回数据库技术。

23.2 闪回数据库

23.2.1 闪回数据库概述

闪回数据库技术是一种快速的数据库恢复方案，这种恢复是基于用户的逻辑错误，比如对表中的数据做了错误的修改、插入了大量错误数据、删除了一个用户等，此时往往需要将数据库恢复

到修改之前的某个时间点。如果是传统的恢复技术，则首先需要复原备份的数据文件，再使用归档日志恢复到以前的某个时间点，显然这样的恢复涉及很多不必要的数据恢复，并且恢复的时间主要取决于数据库的大小。这样为了“小错误”而动用全库的恢复是很耗时的行为。

Oracle 的闪回数据库技术就解决了这个问题，它使用闪回日志来恢复用户的逻辑错误，这种恢复只针对用户逻辑错误的恢复，而不涉及整个数据库的恢复，恢复更具有针对性而且恢复时间大大减少。

闪回日志由 Oracle 自动创建，并存储在闪回恢复区中，由闪回恢复区管理。既然闪回日志由闪回恢复区管理，那么就不能保证闪回日志被保存的数据的可靠性，因为一旦快闪恢复区空间不足，就会自动删除旧的闪回日志文件以腾出空间。在 Oracle 中，快闪恢复区中备份文件的存储优先，所以为了保存尽可能多的闪回日志数据，最好将闪回恢复区设置得大些。

下面我们给出闪回数据库的架构图，如图 23-1 所示。

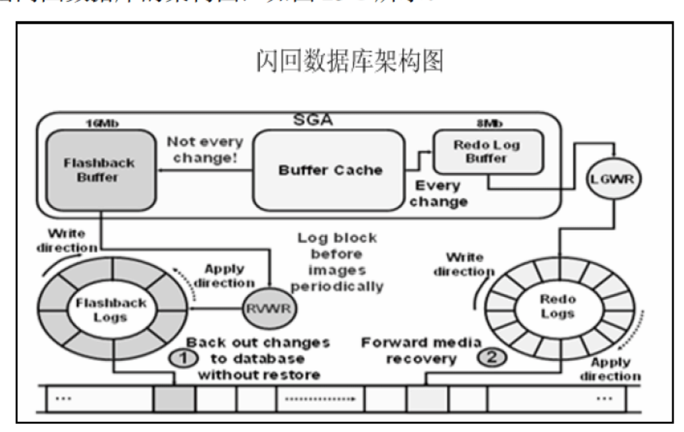


图 23-1　闪回数据库架构图

在图 23-1 中闪回缓冲区中（Flashback Buffer）的变化数据将按照一定的时间间隔，顺序的被写入闪回日志（Flashback Logs），比如用户对表删除或者增加了数据，此时变化的前像操作就记录在闪回日志中，一旦闪回将利用相反的操作恢复修改。注意将闪回缓冲区中的数据写入闪回日志的操作由 RVWR 进程负责，一旦启动了闪回数据库，该进程会自动启动。

显然，如果数据库中绝大多数是查询操作，显然此时的闪回数据库开销很小，因为它不需要做什么，所以闪回数据库的系统开销取决于是否有密集型的写操作。



对于闪回数据库而言，闪回日志不会被归档。

23.2.2　启用闪回数据库

Oracle 默认不启动闪回数据库，如果需要启动闪回数据库特性必须将数据库设置为归档模式，并启用闪回恢复区，因为闪回日志文件存放在闪回恢复区中。如果在 RAC 环境下，必须将闪回恢复区存储在集群文件或 ASM 文件中。按照如下步骤依启用闪回数据库特性。

闪回数据库特性需要数据库处于归档模式，首先检查当前数据库的归档状态，如例子 23-1 所示。

例子 23-1　检查当前数据库的归档状态

```
SQL> archive log list;
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            USE DB RECOVERY FILE DEST
Oldest online log sequence     6
Next log sequence to archive   8
Current log sequence           8
```

显然，当前的数据库处于归档模式，使用 DB_RECOVERY_FILE_DEST 参数指定的目录作为储存目录，该参数的值即为快速恢复区。接着可以继续查询该目录，以确定归档的操作系统存储位置。

例子 23-2　确定归档的操作系统存储位置

```
SQL> show parameter db recovery file Dest;

NAME                         TYPE        VALUE
------------------------------------ -----------------------------
db recovery file dest string          /u01/app/oracle/flash recovery area
db_recovery_file_dest_size           big integer 2G
```

快闪恢复区为/u01/app/oracle/flash_recovery_area 而系统为该空间分配的大小为 2G，这里我们就使用该空间作为系统的归档目录，也作为闪回数据库日志的存储空间。

如果数据库没有启动归档模式，则需要手工启动到归档模式。具体方法为启动数据库到 MOUNT 状态，然后使用 ALTER DATABASE ARCHIVELOG 启动归档模式，如例子 23-3 所示。

例子 23-3　启动归档模式

```
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1350380 bytes
Variable Size             136803636 bytes
Database Buffers          448790528 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
SQL> alter database archivelog;

数据库已更改。
```

设置参数 DB_FLASHBACK_RETENTION_TARGET，该参数的值是一个以分钟为单位的数字，默认为 1440 分钟，含义上将数据库闪回到过去的时间，即从当前开始计算最大可以把数据库闪回到过去的时间。我们可以查询该参数的设置，如例子 23-4 所示。

例子 23-4　查询 DB_FLASHBACK_RETENTION_TARGET 参数的设置

```
SQL> show parameter db flashback retention;

NAME                                 TYPE        VALUE
------------------------------------------------ ------------------
db_flashback_retention_target        integer     1440
```

上述查询的值是系统默认的参数值，可以动态修改这个参数，使得闪回数据库可以闪回更长时间的数据，比如可以闪回到过去两天内的数据，即 24×60×2=2880，如例子 23-5 所示。

例子 23-5　修改 DB_FLASHBACK_RETENTION_TARGET 参数

```
SQL> alter system set db flashback retention target=2880 scope =both;

System altered.
```

参数 DB_FLASHBACK_TETENTION_TARGET 说明可以闪回数据库到过去的时间段，但是必须明确 Oracle 不保证一定可以闪回到时间段内的某个时间点，因为闪回日志是由快闪恢复区自动维护的，如果由于备份数据库而空间不足，此时较早的闪回日志会被删除。为了尽量避免这种情况，在系统运行一段时间，在这段时间内覆盖了数据库系统的主要工作负荷，这样通过数据字典 v$flashback_database_log 来评估需要的快闪恢复区空间，如例子 23-6 所示。

例子 23-6　通过数据字典 v$flashback_database_log 来评估需要的快闪恢复区空间

```
SQL> select estimated_flashback_size,retention_target,flashback_size
  2  from v$flashback database log;

ESTIMATED FLASHBACK SIZE RETENTION TARGET FLASHBACK SIZE
--------------------------------------- ------------------------------
              107249664              1340      40386560
```

此时，参数 ESTIMATED_FLASHBACK_SIZE 说明系统估计的快闪恢复区大小为 107 249 664 字节。FLASHBACK_SIZE 说明当前的闪回数据的大小为 40 386 560 字节。

启动闪回数据库。要启用闪回数据库使用 ALTER DATABASE FLASBHBACK ON 指令，但是该指令必须在数据库处于 MOUNT 状态时运行，先关闭数据库并启动到 MOUNT 状态，然后启用闪回数据库。

例子 23-7　启动数据库到 MOUNT 状态

```
SQL> startup mount;
ORACLE instance started.

Total System Global Area  285223672 bytes
Fixed Size                 2318992 bytes
Variable Size             71304784 bytes
Database Buffers         209715200 bytes
Redo Buffers               2973696 bytes
Database mounted.
```

我们将数据库启动到 MOUNT 状态，如果此时数据库处于打开状态，而用户又试图启动闪回数据库，则会报如下错误。

```
SQL> alter database flashback on;
alter database flashback on
*
ERROR at line 1:
ORA-38759: Database must be mounted by only one instance and not open.
```

接着，在数据库处于 MOUNT 状态时，启动闪回数据库，如例子 23-8 所示。

例子 23-8　启动闪回数据库

```
SQL> alter database flashback on;

Database altered.
```

在数据库启动了闪回特性后，可以通过数据字典视图 V$DATABASE 查看启用状态，如例子 23-9 所示。

例子 23-9　通过数据字典视图 V$DATABASE 查看启用状态

```
SQL> select dbid ,name ,flashback on from v$database;

     DBID       NAME      FLASHBACK_ON
-------------------------------- -------------------------------------
2059883796  TEST       YES
```

显然，在 TEST 数据库上启动了闪回数据库特性，因为 FLASHBACK_ON 列的值为 YES。

23.2.3　关闭闪回数据库

默认情况下，只要启动了闪回数据库特性，数据库的永久表空间都会受闪回数据库保护，如果不希望某个表空间受闪回数据保护，可以禁用对某个表空间的闪回特性，如例子 23-10 所示。

例子 23-10　禁用对某个表空间的闪回特性

```
SQL> alter tablespace users flashback off;

system changed。
```

我们通过数据字典 V$TABLESPACE 来查询该表空间是否已经不被闪回保护，如例子 23-11 所示。

例子 23-11　查询该表空间是否已经不被闪回保护

```
SQL> col flashback_on for a15
SQL> select name,flashback_on
  2* from v$tablespace

NAME                                       FLASHBACK ON
------------------------------------ -------------------------------
SYSTEM                                       YES
UNDOTBS1                                     YES
SYSAUX                                       YES
USERS                                        NO
TEMP                                         YES
TEST                                         YES

已选择 6 行。
```

从输出可以看出表空间 USERS 已经不被闪回数据库保护了。如果想重新启用该表空间，使其

继续启用闪回数据库特性，可以通过指令 ALTER TABLESPACE USERS FLASHBACK ON 来启用，但是必须将数据库启动到 MOUNT 状态。

例子 23-12　将数据库启动到 MOUNT 状态后，启用闪回数据库特性

```
SQL> startup mount;
ORACLE 例程已经启动。
…………
数据库装载完毕。
SQL> alter tablespace users flashback on;

表空间已更改。
```

此时，我们继续使用数据字典 V$TABLESPACE 来查询当前数据库表空间被闪回数据库保护的状态，如例子 23-13 所示。

例子 23-13　使用数据字典 V$TABLESPACE 来查询被闪回数据库保护的状态

```
SQL> select name,flashback_on
  2* from v$tablespace

NAME                                            FLASHBACK_ON
----------------------------------------------- ---------------
SYSTEM                                          YES
UNDOTBS1                                        YES
SYSAUX                                          YES
USERS                                           YES
TEMP                                            YES
TEST                                            YES

已选择 6 行。
```



此时的表空间 USERS 的 FLASHBACK_ON 为 YES，说明它又重新被闪回数据库保护了。

如果需要关闭闪回数据库特性，可以在关闭数据库级别的闪回数据库特性，如下所示。

例子 23-14　关闭闪回数据库特性

```
SQL> startup mount;
ORACLE 例程已经启动。
…………
数据库装载完毕。
SQL> alter database flashback off;

数据库已更改。
```



上述操作必须将数据库实例启动到 MOUNT 状态，否则无法关闭或启动闪回数据库特性，并且一旦关闭闪回数据库特性，闪回日志将自动删除。

23.2.4 闪回数据库方法

闪回数据库可以使用 RMAN 方法也可以使用 SQL 指令的方法实现，使用 RMAN 闪回数据库有如下 3 种方法。

```
RMAN>FLASHBACK DATABASE TO TIME=TO DATE('2010-5-25 23:43:00',
         'YYYY-MM-DD HH23:MI:SS');
```

该方法将数据库闪回到过去的某个时间点，通过 TO_DATE 函数指定具体的时间。

```
RMAN>FLASHBACK DATABASE TO SCN=638832;
```

该方法将数据库闪回到过去的某个系统 SCN，显然在实践中这个方法不容易实现，因为在数据库发生逻辑错误之前一般不会去查询数据库当前的 SCN。

```
RMAN>FLASHBACK DATABASE TO SEQUENCE=345 THREAD=1;
```

该方法闪回到特定日志序列号之前的状态，不包括序列号 345。

使用 SQL 指令闪回数据库有如下两种方式。

```
SQL>FLASHBACK DATABASE TO TIMESTAMP(SYSDATE-1/24);
```

该法方法将数据库闪回到时间戳指定的状态。

```
SQL>FLASHBACK DATABASE TO SCN 678854
```

该方法闪回数据库到过去的某个 SCN。

在执行闪回数据库时，需要将数据库切换到 MOUNT 状态，在闪回数据库结束后，必须使用 ALTER DATABASE OPEN RESETLOGS 打开数据库，即需要重新设置重做日志，使得重做日志序列号重新计数。

23.2.5 使用闪回数据库

本节的目的是通过一个例子演示如何闪回用户的错误操作，我们创建一个用户，并使用该用户创建一些表，然后插入数据并提交，最后通过删除该用户以及其包含的所有数据来模拟一个逻辑错误，然后通过闪回数据库恢复。

首先，创建用户 VFAST。

例子 23-15 创建用户 VFAST

```
SQL> create user vfast identified by vfast account unlock;

User created.

SQL> grant resource,connect to vfast;

Grant succeeded.
```

创建一个表并插入数据。

例子 23-16　创建一个表并插入数据

```
SQL> create table vtest (id number,name varchar2(20));

Table created.

SQL> insert into vtest values (1,'name1');

1 row created.

SQL> commit;
```

我们继续查看此时的系统时间，在闪回时我们就闪回数据库到这个时刻。

例子 23-17　查看此时的系统时间

```
SQL> select to_char(sysdate,'yyyy-mm-dd hh24:mi:ss') from dual;

TO_CHAR(SYSDATE,'YY
-------------------
2011-09-20 10:38:22
```

删除用户 VFAST。

例子 23-18　删除用户 VFAST

```
SQL> drop user vfast cascade;

User dropped.
```

在删除了 VFAST 之后，在闪回日志中就记录了相关数据，我们通过动态数据字典视图 v$flashback_database_log 来查询。

例子 23-19　查询动态数据字典视图 v$flashback_database_log

```
SQL> select * from v$flashback_database_log;

OLDEST_FLASHBACK_SCN OLDEST_FL RETENTION_TARGET FLASHBACK_SIZE
-------------------- --------- ---------------- --------------
ESTIMATED_FLASHBACK_SIZE
------------------------
 633118               20-SEP-11    2880             8192000
 238583808
```

在该视图中记录闪回日志区域可以闪回到的最早 SCN 以及最早时间，并且评估了所需要闪回恢复区的大小。下面我们查询可以闪回到的最早时间。

例子 23-20　查询可以闪回到的最早时间

```
SQL> select to_char(oldest_flashback_time,'yyyy-mm-dd hh24:mi:ss') from
v$flashback_database_log;

TO_CHAR(OLDEST_FLAS
-------------------
2011-09-20 10:23:26
```

从输出可以看出，可以将数据库最早闪回到 2011-09-20 10:23:26。下面我们将数据库闪回到该时间点之后的时间点 2011-09-20 10:38:22，恢复误删除用户 VFAST。

例子 23-21　通过闪回数据库恢复误删除用户 VFAST

```
SQL> startup mount;
ORACLE instance started.

Total System Global Area  285223672 bytes
Fixed Size                  2318992 bytes
Variable Size              75499088 bytes
Database Buffers          205520896 bytes
Redo Buffers                2973696 bytes
Database mounted.
SQL> flashback database to timestamp to_date('2011-09-20 10:38:22','yyyy-mm-dd hh24:mi:ss');
Flashback complete.
```

提示闪回完成，这时我们最好使用 READ ONLY 模式打开数据库，先验证所需要的数据是否成功恢复，如果没有恢复可以继续使用闪回数据库恢复，直到确定已经成功恢复到指定的时间点。

验证数据的正确性：

例子 23-22　使用 READ ONLY 模式打开数据库

```
SQL> alter database  open read only;

Database altered.
```

验证数据的正确性：

例子 23-23　验证数据的正确性

```
SQL> select username,account_status from dba_users where username like 'VF%';

USERNAME                       ACCOUNT_STATUS
------------------------------ --------------------------------
VFAST                          OPEN

SQL> select * from vfast.vtest;
        ID NAME
---------- --------------------
         1 name1
```

通过上面的查询，我们确定已经闪回到合适的时间点后，再使用 RESETLOGS 打开数据库。

```
SQL> startup mount;
ORACLE instance started.

Total System Global Area  285223672 bytes
Fixed Size                  2318992 bytes
Variable Size              75499088 bytes
Database Buffers          205520896 bytes
```

```
Redo Buffers                2973696 bytes
Database mounted.
SQL> alter database open resetlogs;

Database altered.
```

使用 resetlogs 打开数据库，闪回日志仍然有效。还可以继续闪回到 resetlogs 以前的某一个时间点。也可以闪回数据库到某个 SCN，如下所示。

```
SQL>flashback database to scn 633118;
```

使用 rman 也可以闪回数据库。但是 rman 中不能使用 to_char 和 to_date，因此需要提前设置 NLS_DATE_FORMAT 以及 NLS_LANG。



如果使用闪回操作造成了一定的混乱，比如闪回没有达到要求，可以使用 RECOVER DATABASE 撤销闪回操作。如果闪回太多，可以使用 RECOVER DATABASE UNTIL 将数据库恢复到以前的某个时间。

23.2.6　监控闪回数据库

本节我们介绍几个监控闪回数据库的数据字典视图，通过它们可以知道将数据库闪回到过去的时间以及 SCN，当前闪回日志的各种状态，如闪回起止时间、闪回记录的数据量等信息。下面我们分别介绍这些视图。

因为无法保证一定可以闪回到从现在开始到参数 db_flashback_retention_target 指定时间段内的任意时间点，所以在闪回前最好使用数据字典 V$FLASHBACK_DATABASE_LOG 来查询可以闪回到的最小 SCN 号以及可以闪回到的时间点，如例子 23-24 所示。

例子 23-24　查询可以闪回到的最小 SCN 号以及可以闪回到的时间点

```
SQL> select oldest_flashback_scn,oldest_flashback_time
  2  from v$flashback_database_log;

OLDEST_FLASHBACK_SCN OLDEST_FLASHBA
-------------------- --------------
              613813    25-5 月 -10
```

输出说明，可以将当前数据库闪回到的最小的 SCN 为 613813。此时，我们可以继续查看系统当前的 SCN，可以预知当前 SCN 一定会大于 OLDEST_FLASHBACK_SCN 参数的值。

例子 23-25　查看系统当前的 SCN

```
SQL> select current_scn
  2  from v$database;

CURRENT_SCN
-----------
     613155
```

显然，当前的 SCN 为 613155，比 OLDEST_FLASHBACK_SCN 参数的值要大。下面我们介

绍数据字典视图 V$FLASHBACK_DATABASE_STAT，该字典视图的作用是监视闪回日志写入闪回数据的各种开销，如记录了当前闪回记录起止时间，闪回记录的数据量以及重做日志记录的数据量等信息。该视图记录 24 小时内的闪回日志开销记录，每一行记录了一小时间隔的状态，如例子 23-26 所示。

例子 23-26　视图 V$FLASHBACK_DATABASE_STAT 记录了 24 小时内的闪回日志开销

```
    SQL>SELECT *
      2  FROM V$FLASHBACK_DATABASE_STAT;

    BEGIN TIME    END TIME     FLASHBACK DATA            DB DATA        REDO DATA
ESTIMATED FLASHBACK SIZE
    ---------------------------------------------------------------------------
    25-5 月 -10    25-5 月 -10            835584     598016     151040        0
```

其中 FLASHBACK_DATA 表示在时间间隔内记录的多少闪回数据，单位是字节参数 DB_DATA 记录了时间间隔内有多少数据块的读写，单位是数据块。参数 REDO_DATA 说明时间间隔内记录了多少重做数据，单位是字节。

我们已经反复强调过，闪回数据库不保证闪回数据到过去的某个时间点，因为闪回日志是由快闪恢复区维护的，而快闪恢复区是备份文件优先存储，即如果由于备份的需要空间不足，就会删除部分闪回日志文件。所以虽然定义了参数 db_flashback_retention_target，但是 Oracle 只是尽力保证恢复到该参数指定的从现在开始某个时间段内的数据。

所以需要监控快闪恢复区的空间变化，在必要时候增加快闪恢复区的空间，如例子 23-27 所示。

例子 23-27　增加快闪恢复区的空间

```
    SQL> select name,space_limit,space_used,space_reclaimable,number_of_files
      2* from v$recovery_file_dest

    NAME         SPACE LIMIT SPACE USED SPACE RECLAIMABLE NUMBER OF FILES
    ---------------------------------------------------------- ---------------
    E:\oracle\product\10.2.0/flash_recovery_area       2337483648     669641728
55032832    8
```

参数 NAME 说明了快闪恢复区的目录，参数 SPACE_LIMIT 说明空间最大使用上限，参数 SPACE_USED 说明已经使用了的空间，参数 SPACE_RECLAIMABLE 说明可以回收的空间，此时需要注意已经使用的空间和总空间的值，如果二者接近就增加闪回恢复区的尺寸，或者使用其他方式，比如闪回恢复区管理等方法，以防止快闪恢复区空间不足。

23.2.7　使用闪回数据库的限制

在以下几种环境下不能使用闪回数据库特性。

- 如果是数据文件被删除或缩短。
- 如果在闪回时间范围内复原或重建了一个控制文件。
- 在 RESETLOGS 操作之前。
- 表空间被删除。

在以上几种情况无法使用闪回数据库时，只能使用不完全恢复将数据库恢复到过去的某个时间点。

23.3　闪回删除

闪回删除的目的是防止用户错误地删除了表、索引等数据库对象，在未使用闪回技术之前，只能使用传统的数据恢复方式从备份中恢复，此时需要备份文件以及归档日志文件，这样的恢复往往涉及那些不需要恢复的对象，显然这样的恢复不具有针对性，而使用闪回删除就可以直接恢复想恢复的对象，更高效省时。

启动闪回后，在删除一个表时，物理上该表没有被删除，但是它所占用的空间回到空闲列表，也就是这段空间在某种条件下是可以被占用的。

23.3.1　闪回删除原理

如果使用 DROP TABLE 指令删除表，该表不会从数据库中立即删除，而是保持原表的位置，但是将删除的表重新命名，并将删除的表信息存储在回收站中，回收站记录了被删除表的新名字和原名字。显然此时被删除的表所占有的空间没有被立即释放，变成数据库可以使用的潜在空间。记录在回收站中的信息会保留一段时间，直到回收站空间不足或者使用 PURGE 指令删除回收站中的记录。

回收站是一个逻辑结构，不具有物理数据结构，只要删除的表信息记录在回收站中就可以通过闪回技术恢复删除的表。

下面我们查看回收站的相关信息，如图 23-2 所示。

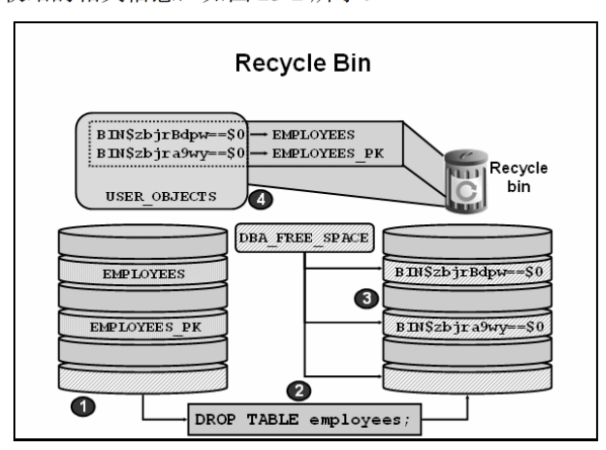


图 23-2　闪回删除原理图

如图 23-2 所示，第一步删除表 employees，然后该表对应的物理数据块空间就成为备用的可用空间，该删除记录保存记录在回收站中，回收站为删除的表重新命名并记录该表的原始名，这样用户就很容易查询删除的表以及对应的原始表名。

要启动闪回删除，关键是启动 recyclebin，可以通过 alter system set 指令动态设置该参数，默

认 Oracle 启动闪回删除，如例子 23-28 所示。

例子 23-28　查询 Oracle 是否启动闪回删除

```
SQL> show parameter recyclebin;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------------
recyclebin                           string      on
```

显然参数 RECYCLEBIN 的值为 ON，所以当前数据库上启动了闪回删除，如果该参数值显示为 OFF，则使用如下所示启动闪回删除。

例子 23-29　启动闪回删除

```
SQL> alter system set recyclebin=on scope=both;

系统已更改。
```

23.3.2　回收站的使用

在闪回删除原理中，我们已经理解了回收站其实是一个逻辑结构，记录了被删除表的逻辑信息，Oracle 提供了数据字典视图 USER_RECYCLEBIN 和 DBA_RECYCLEBIN 供用户查询数据库中回收站中记录的信息。

先看数据字典 DBA_RECYCLEBIN 的结构。

```
SQL> desc dba_recyclebin;
 名称                                                   是否为空? 类型
 ----------------------------------------------------- -------- ----------------

 OWNER                                     NOT NULL    VARCHAR2(30)
 OBJECT_NAME                               NOT NULL    VARCHAR2(30)
 ORIGINAL_NAME                                         VARCHAR2(32)
 OPERATION                                             VARCHAR2(9)
 TYPE                                                  VARCHAR2(25)
 TS_NAME                                               VARCHAR2(30)
 CREATETIME                                            VARCHAR2(19)
 DROPTIME                                              VARCHAR2(19)
 DROPSCN                                               NUMBER
 PARTITION_NAME                                        VARCHAR2(32)
 CAN_UNDROP                                            VARCHAR2(3)
 CAN_PURGE                                             VARCHAR2(3)
 RELATED                                   NOT NULL    NUMBER
 BASE_OBJECT                               NOT NULL    NUMBER
 PURGE_OBJECT                              NOT NULL    NUMBER
 SPACE                                                 NUMBER
```

由于在闪回删除中回收站的重要作用，我们将详细解释该数据字典的结构。

- OWNER：表示被删除的表所属用户。
- OBJECT_NAME：为 Oracle 为删除的表的重命名。

- ORIGINAL_NAME：为被删除表的原始表名。
- OPERATION：对表的操作，因为是删除表所以该列会显示为 DROP。
- TYPE：被删除的数据库对象类型，如表或索引等。
- TS_NAME：被删除的数据库对象对应的表空间。
- CREATETIME：回收站中被删除对象的创建时间。
- DROPTIME：删除时间。
- CAN_UNDROP：记录对象是否可以闪回删除。
- CAN_PURGE：该记录是否可以被永久删除。

下面通过一个例子充分理解闪回删除回收站的记录信息。先创建一个测试表，然后删除该表，查看该表的回收站中的记录。先创建一个表，如例子 23-30 所示。

例子 23-30　先创建一个表

```
SQL> create table scott.emp test
  2  as
  3  select *
  4  from scott.emp;

表已创建。
```



表 emp_test 属于 SCOTT 用户，该信息也会记录在回收站中。

例子 23-31　删除表 emp_test

```
SQL> drop table scott.emp_test;

表已删除。
```

我们模拟了一个用户错误地删除了表 EMP_TEST，如果是传统的恢复方式则需要使用包含该表的备份文件，并且恢复过程时间相对较长也涉及一些不必要的数据的恢复，因为使用了闪回技术，此时虽然删除了该表，但是该表的数据还是保留在原处，此时仅仅将该表从数据字典中删除记录。

下面我们查看回收站中该表的记录，在回收站中记录了该表的原始名称和回收站中的名称，即对象名。

我们使用 SYS 用户登录数据库，查询回收站中记录的删除表的信息，如例子 23-32 所示。

例子 23-32　查询回收站中记录的删除表的信息

```
SQL> col ts name for a10
SQL> col owner for a10
SQL> col original_name for a10
SQL>select owner,original_name,object_name,ts_name,droptime
  2* from dba recyclebin

OWNER   ORIGINAL_N OBJECT_NAME                    TS_NAME  DROPTIME
---------- ---------- ------------------ ---------- -------------------
SCOTT    TEST      BIN$y181Im8wRXS9kBVBJ60uTA==$0 USERS   2010-05-22:22:49:45
SCOTT    EMP_TEST  BIN$2bqYJuA2QBG3hIePCE9Dcg==$0 USERS   2010-05-25:11:57:18
```

从输出可以看出，表 EMP_TEST 已经记录在回收站中，记录该表数据 SCOTT 用户，所属表空间为 USERS，并记录了删除时间为 2010-05-25:11:57:18，注意此时自动生成了一个被删除表的系统名称为 BIN$2bqYJuA2QBG3hIePCE9Dcg==$0，该名称由 30 个字符组成。



在重命名被删除的表时，也会重命名该表的依赖对象，如涉及的索引、触发器等，在恢复表时，该表涉及的依赖对象会自动恢复，但是会保留系统名称。

我们也可以使用数据字典 USER_RECYCLEBIN 以及 RECYCLEBIN 查询用户回收站中删除的数据库对象信息。此时的 RECYCLEBIN 是 USER_RECYCLEBIN 的同义词。但是，此时必须在对象所属的用户模式下，如我们删除的表 EMP_TEST 是 SCOTT 用户拥有的表，所以必须使用 SCOTT 用户模式查询 RECYCLEBIN，如例子 23-33 所示。

例子 23-33　在 SCOTT 用户模式下查询 RECYCLEBIN

```
SQL> conn scott/oracle
已连接。
SQL> show recyclebin;
ORIGINAL NAME  RECYCLEBIN NAME              OBJECT TYPE  DROP TIME
---------------------------------------------------------------------------
EMP TEST     BIN$2bqYJuA2QBG3hIePCE9Dcg==$0  TABLE     2010-05-25:11:57:18
TEST         BIN$y18lIm8wRXS9kBVBJ60uTA==$0  TABLE     2010-05-22:22:49:45
```



如果使用 USER_RECYCLEBIN 查询以上输出的数据，结果相同，其实 USER_RECYCLEBIN 比 DBA_RECYCLEBIN 缺少一个 OWNER 属性，其他二者相同。

23.3.3　恢复删除的表

要删除回收站中的表，必须使用在该表所属的用户模式下操作闪回操作，如果删除了用户 SCOTT 的表 EMP_TEST，需要在 SCOTTT 用户模式下实现闪回操作，使用 FLASHBACK TABLE…TO BEFORE DROP，其实就是要求用户具有闪回操作的权限，如具有 DROP TABLE 等权限，读者需要注意，如果操作中出现“权限限制”，首先需要赋予该用户相对应的操作权限。如例子 23-34 所示，我们恢复删除的表 EMP_TEST。

例子 23-34　恢复删除的表 EMP_TEST

```
SQL> flashback table emp_test to before drop;

闪回完成。
```

此时，我们继续看回收站中是否还有该表记录，如例子 23-35 所示。

例子 23-35　验证回收站中是否还有表 EMP_TEST 的记录

```
SQL> show recyclebin;
ORIGINAL NAME    RECYCLEBIN NAME            OBJECT TYPE  DROP TIME
-------------------------------------- ------------ -------------------
TEST            BIN$y18lIm8wRXS9kBVBJ60uTA==$0 TABLE     2010-05-22:22:49:45
```

显然，回收站中已经没有了表 EMP_TEST 的记录，我们继续查询表 EMP_TEST 的信息，如例子 23-36 所示。

例子 23-36　查询表 EMP_TEST 的信息

```
SQL> select table name
  2  from user tables;

TABLE_NAME
------------------------------
DEPT
EMP
BONUS
SALGRADE
EMP TEST

已选择 6 行。
```

显然，表 EMP_TEST 已经重新记录在数据字典 DBA_TABLES 中了，下面我们再继续查看表 EMP_TEST 中的数据，如例子 23-37 所示。

例子 23-37　查看表 EMP_TEST 中的数据

```
SQL> select *
  2  from emp test
  3  where rownum<2;

    EMPNO  ENAME      JOB         MGR  HIREDATE      SAL  COMM  DEPTNO
---------------- ---------------------- ---------- ---------- ----------
     7369   SMITH      CLERK       7902  17-13月-80     800          20
```

从输出显而易见，表 EMP_TEST 的数据也已经恢复，这样我们通过闪回删除技术成功恢复了用户“误删除”的表 EMP_TEST。

在闪回表时，也可以使用系统为被删除表的重命名后的名称，并且可以为恢复后的表重新命名。我们先查询当前用户 SCOTT 回收站中的记录。

例子 23-38　查询当前用户 SCOTT 回收站中的记录

```
SQL> show recyclebin;
ORIGINAL NAME    RECYCLEBIN NAME              OBJECT TYPE  DROP TIME
---------------------------- ------------------------ --------------------
TEST        BIN$y181Im8wRXS9kBVBJ60uTA==$0  TABLE       2010-05-22:22:49:45
```

从输出发现表 TEST 被删除，并且被重命名为 BIN$y18lIm8wRXS9kBVBJ60uTA==$0，我们通过系统名称 BIN$y18lIm8wRXS9kBVBJ60uTA==$0 恢复表 TEST，并且重命名为 NEW_TEST，如例子 23-39 所示。

例子 23-39　恢复表 TEST 并重命名为 NEW_TEST

```
SQL> flashback table "BIN$y181Im8wRXS9kBVBJ60uTA==$0"
  2  to before drop
  3  rename to new_test;
```

```
闪回完成。
```

提示闪回完成，我们验证闪回结果，注意此时已经将表 TEST 重命名为 NEW_TEST，如例子 23-40 所示。

例子 23-40　验证闪回结果

```
SQL> select table_name
  2  from user_tables;

TABLE_NAME
------------------------------
DEPT
EMP
BONUS
SALGRADE
EMP_TEST
NEW_TEST

已选择 6 行。
```

此时，我们发现表 NET_TEST 已经重新注册到数据字典中，读者可以查询表 NEW_TEST 中的数据以验证数据的恢复，这里不再演示。

我们讲过一旦 DROP 一个表，和表相关联的其他数据库对象如触发器、索引都将被删除；一旦闪回该表时，关的数据库对象将自动恢复，但是名称不是原先的名称，而是删除后系统自动重起的名称，该名称不易阅读，需要手工修改。下例我们演示这个过程，首先创建一个表 TEST1，并创建一个索引，然后删除表再闪回该表查看索引变化。

例子 23-41　创建一个表 TEST1

```
SQL> create table test1
  2  as
  3  select *
  4  from emp;

表已创建。
```

创建基于表 TEST1 的列 ENAME 的索引 SCOTT_TEST1_ENAME，如例子 23-42 所示。

例子 23-42　创建基于表 TEST1 的列 ENAME 的索引

```
SQL> create index scott_test1_ename on test1(ename);

索引已创建。
```

接着，验证该索引的创建结果，如例子 23-43 所示。

例子 23-43　验证该索引的创建结果

```
SQL> select index_name,table_name
  2  from user_indexes
  3  where table_name='TEST1';
```

```
INDEX_NAME                        TABLE_NAME
------------------------------ ---------------------------
SCOTT_TEST1_ENAME                  TEST1
```

现在我们成功创建了一个表 TEST1，并基于该表的 ENAME 列创建了索引 SCOTT_TEST1_ENAME，现在我们要先删除该表，再闪回该表，查看基于该表的索引名称的变化。

例子 23-44　先删除表 TEST1，再闪回该表

```
SQL> drop table test1;

表已删除。
```

我们通过同义词 RECYCLEBIN 查看回收站中是否记录了表 TEST1 的删除记录，如例子 23-45 所示。

例子 23-45　查看回收站中是否记录了表 TEST1 的删除记录

```
SQL> show recyclebin;
ORIGINAL NAME    RECYCLEBIN NAME          OBJECT TYPE  DROP TIME
---------------- ------------------------------------- -------------------
TEST1                              BIN$1z+DpGAGQquDxt0ygK4B3g==$0    TABLE
2010-05-25:15:34:53
```

显然，表 TEST1 的删除结果记录在回收站中。下面我们查询该表的索引 SCOTT_TEST1_ENAME 是否存在，如例子 23-46 所示。

例子 23-46　查询该表的索引 SCOTT_TEST1_ENAME 是否存在

```
SQL>  select index_name,table_name
  2   from user_indexes
  3   where table name='TEST1';

未选定行
```

显然，此时数据字典 USER_INDEXES 已经删除了索引记录，我们继续闪回表 TEST1，如例子 23-47 所示。

例子 23-47　闪回表 TEST1

```
SQL> flashback table test1 to before drop;

闪回完成。
```

按照闪回原理，与表 TEST1 相关联的数据库对象也会自动闪回，下面我们通过数据字典 USER_INDEXES 查询表 TEST1 的索引是否闪回成功，如例子 23-48 所示。

例子 23-48　查询表 TEST1 的索引是否闪回成功

```
SQL> select index_name,table_name
  2  from user indexes
  3  where table name='TEST1';
```

```
INDEX NAME                     TABLE NAME
--------------------------------------------------------
BIN$1PD6fSonT26sNyykRvaf3g==$0    TEST1
```

我们看到表 TEST1 的索引闪回成功，但是名称依然是在闪回表时为该索引起的名字，所以需要用户自己修改该索引的名字。

例子 23-49　修改该索引的名字

```
SQL> drop index "BIN$1PD6fSonT26sNyykRvaf3g==$0" ;

索引已删除。
SQL> create index scott_test1_ename on test1(ename);

索引已创建。
```

显然，我们是通过先删除索引 BIN$lPD6fSonT26sNyykRvaf3g==$0，然后再重建以前的索引的方式重建索引。这样在闪回表后就重命名了闪回表的相关数据库对象。



如果不知道索引的定义，可以使用 dba_metadata.get_ddl 函数获得数据库对象定义，然后在闪回表后重建对应的索引。

23.3.4　恢复多个同名的表

在生产库中，有可能同名的表都被删除，并且这些删除的表都记录在回收站中而没有被清除，那么如何使用闪回来恢复同名的表呢？此时我们需要确认回收站中的哪个表是我们需要闪回的。读者需要注意启动闪回后表实际上还是存在的，只是名字变了，我们同样可以使用 DESC 指令确认该表的结构，或者查看删除时间等方法。下面我们模拟这个过程。

首先创建一个表 FTEST。

例子 23-50　创建一个表 FTEST

```
SQL> create table ftest (id number);

Table created.

SQL> insert into ftest values(1000);

1 row created.

SQL> commit;

Commit complete.
```

然后，删除该表。

例子 23-51　删除表 FTEST

```
SQL> drop table ftest;

Table dropped.
```

接着，我们创建同名的表 FTEST，但是二者的结构不同。

例子 23-52　创建同名的表 FTEST

```
SQL> create table ftest (id number,name varchar2(20));

Table created.

SQL> insert into ftest values (1,'name1');

1 row created.

SQL> commit;

Commit complete.
```

我们再次删除该表。

例子 23-53　再次删除表 FTEST

```
SQL> drop table ftest;

Table dropped.
```

此时，我们删除了同名的两个表，此时 Oracle 将两个表重新命名放入回收站，这两个表段的空间成为潜在的回收对象。下面我们查询回收站中记录的信息。

例子 23-54　查询回收站中记录的信息

```
SQL> show recyclebin;
ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
---------------------------------------------- ------------ -------------------
FTEST          BIN$rVgfOy5oLVTgQKjACgEOyg==$0 TABLE        2011-09-20:23:04:59
FTEST          BIN$rVgfOy5nLVTgQKjACgEOyg==$0 TABLE        2011-09-20:23:02:49
```

此时，回收站中记录了被删除的两个表，虽然原始名称相同，但是毕竟是两个不同的数据库对象，在回收站中重命名后都有各自的名字，此时，如果需要闪回具有两个列属性的表 FTEST，我们就需要使用 DESC 指令查看表的结构，如例子 23-55 所示。

例子 23-55　使用 DESC 指令查看表的结构

```
SQL> desc "BIN$rVgfOy5nLVTgQKjACgEOyg==$0";
 Name                              Null?    Type
 ------------------------------------------------- --------------------
 ID                                         NUMBER

SQL> desc "BIN$rVgfOy5oLVTgQKjACgEOyg==$0";
 Name                              Null?    Type
 ----------------------------------------- ----------------------------
 ID                                         NUMBER
 NAME                                       VARCHAR2(20)
```

此时我们知道表"BIN$rVgfOy5oLVTgQKjACgEOyg==$0"对应的原始表 FTEST 是需要闪回的表。如下我们闪回该表并且重命名为 NEW_FTEST。

例子 23-56　闪回表 FTEST 并重命名为 NEW_FTEST

```
    SQL> flashback table "BIN$rVgfOy5oLVTgQKjACgEOyg==$0" to before drop rename
to new_ftest;

    Flashback complete.
```

23.3.5　应用 Purge 永久删除表

我们知道启动闪回后被删除的对象没有被物理地释放，Oracle 提供了两种方式来回收这样的物理空间。

一种是自动回收，即当表空间出现压力时，Oracle 会首先使用表空间里不属于回收站的对象所占用的可用空间，如果这部分空间用完，仍然存在空间压力，则释放回收站里面最早的那些对象所占用的空间。直至释放完毕所有的空间，然后扩展数据文件，前提是数据文件支持自动扩展。

第二种方式就是使用手工方式，使用 purge 指令手工删除回收站里的对象，在实际的数据库维护中，用户确认不需要某个表，又出于该表的安全性所以不希望放入回收站，此时我们使用 PURGE 指令来删除该表即可，如例子 23-57 所示。

例子 23-57　使用 PURGE 指令来删除该表

```
    SQL> drop table test1 purge;

    表已删除。
```

这样删除的表不会在回收站中记录任何信息，如果此时查看回收站不会发现表 TEST1 的删除记录。

有时需要永久删除一个表空间，但是由于该表空间数据量太大，所以不希望该表空间继续占用潜在的磁盘空间，不想放入回收站，此时可以使用 DROP TABLESPACE...INCLUDING CONTENTS 指令实现，如例子 23-58 所示。

例子 23-58　永久删除一个表空间

```
    SQL> drop tablespace test
      2  including contents;

    表空间已删除。
```



上面删除了表空间 test，同时会触发一个操作，即 Oracle 会将回收站中和表空间 TEST 相关的所有表同时删除。

如果已经删除一个表，并且该表记录在回收站中，此时又希望永久删除该表，也需要使用 PURGE 指令。我们先查看当前回收站中的信息以确认要永久删除的表，如例子 23-59 所示。

例子 23-59　查看当前回收站中的信息以确认要永久删除的表

```
    SQL> show recyclebin;
    ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
    ---------------------------- ---------------------- --------------------
```

```
TEST2         BIN$NizZ+ImCQOOd7bTix4x1pA==$0 TABLE       2010-05-25:16:05:58
TEST3         BIN$8by8z0a0RHSOp6YxznMQ6Q==$0 TABLE      2010-05-25:16:06:09
```

此时，如果要永久删除表 TEST2，则使用 PURGE 指令，如例子 23-60 所示。

例子 23-60　永久删除表 TEST2

```
SQL> purge table test2;

表已清除。
```

此时表 TEST2 将从数据库中永久删除，并且和该表对应的其他数据库对象也将永久删除。我们继续查找回收站，则不会再记录表 TEST2 的任何信息，如例子 23-61 所示。

例子 23-61　查找回收站验证

```
SQL> show recyclebin;
ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
---------------------------------------- ------------ --------------------
TEST3       BIN$8by8z0a0RHSOp6YxznMQ6Q==$0 TABLE       2010-05-25:16:06:09
```

此时，回收站中确实已经删除了表 TEST2。

在永久删除回收站中的表时，也可以使用系统生成的名字，如例子 23-62 所示。

例子 23-62　永久删除回收站中系统生成的表

```
SQL> purge table "BIN$NizZ+ImCQOOd7bTix4x1pA==$0";

表已清除。
```

在生产数据库的维护中，我们会遇到这种情况，就是已经知道删除了某个表空间中的多个表，又打算永久删除这些表，此时如果一一删除显然会耗费时间，可以使用如下指令直接删除和某个表空间相关的回收站中的表。先查看当前回收站中的表。

例子 23-63　查看当前回收站中的表

```
SQL> select original_name,object_name,ts_name
  2  from user recyclebin;

ORIGINAL  NAME         RECYCLEBIN  NAME                               TS  NAME
------------------------------------------------------------ --------------------
TEST3               BIN$8by8z0a0RHSOp6YxznMQ6Q==$0   USERS
```

当前只有表 TEST3 在表空间中，并且表 TEST3 属于 USERS 表空间，下面通过 PURGE 表空间的方式永久删除回收站中和表空间 USERS 相关的表，如例子 23-64 所示。

例子 23-64　永久删除回收站中和表空间 USERS 相关的表

```
SQL> purge tablespace users;

表空间已清除。
```

此时，继续查看回收站中的表记录，不会在出现和表空间 USERS 相关的表记录了，如例子 23-65 所示。

例子 23-65　查看回收站中的表记录验证

```
SQL> select original name,object name,ts name
  2  from user_recyclebin;
  3  where ts_name='USERS';

未选定行
```

我们也可以永久删除回收站中和某个表空间相关的某个用户的表，此时除了 TABLESPACE 关键字还需要 USER 关键字，如例子 23-66 所示。

例子 23-66　永久删除回收站中和某个表空间相关的某个用户的表

```
SQL> purge tablespace test user scott;

表空间已清除。
```



使用 PURGE 指令时对回收站的操作，使用 DROP 指令是对数据库中的表的操作，这两个操作有一点关联，但是操作的对象不同。在使用时要注意二者和操作对象。

23.4　闪回表

所谓闪回表，就是将表里的数据回退到历史上的某个时间点，例如回退到用户误删除数据之前的时间点，从而将误删除的数据恢复回来，在这个过程中，数据库仍然可用，而且不需要类似于闪回日志一样的额外空间。

闪回表利用 undo 表空间里记录的数据旧映像，如果闪回表所需要的 undo 数据，由于保留的时间超过了初始化参数 undo_retention 所指定的值，从而导致该 undo 数据块被其他事务覆盖，就不能恢复到指定的时间点了。

我们可以指定 undo 表空间的 retention guarantee 选项，来保证闪回的成功。如下所示，我们查询当前 UNDO 相关参数。

例子 23-67　查询当前 UNDO 相关参数

```
SQL> show parameter undo;

NAME                              TYPE        VALUE
------------------------------------ ------------------------------
undo management                         string      AUTO
undo retention                           integer     900
undo_tablespace                          string      UNDOTBS1
```

从输出看出，当前数据库的 UNDO 表空间是 UNDOTBS1，保留时间是 15 分钟，我将保留时间修改为 24 小时，但是要保证闪回的成功，我们还必须设置 UNDO 表空间的 RETENTION GUARANTEE 选项，如例子 23-68 所示。

例子 23-68　设置 UNDO 表空间的 RETENTION GUARANTEE 选项

```
SQL> alter system set undo_retention=86400 scope=spfile;
```

```
System altered.

SQL> alter tablespace undotbs1 retention guarantee;

Tablespace altered.
```

参数 undo_retention 的作用是 UNDO 中的数据至少保留的时间，在这段时间内不能覆盖，这样就可以保证将表恢复到这个时间段内的某个时间点。

闪回表的操作会修改表里的数据，从而可能引起数据行的移动，比如某一行数据当前在 A 数据块里面，而在表闪回到以前的某个时间点上时，在那个时间点上，该行数据位于 B 数据块里面，因此在闪回表之前，必须启用数据行的移动特性。我们通过一个具体的例子学习这些知识点。

首先创建一个表并插入数据库。

例子 23-69　创建一个表并插入数据库

```
SQL> create table tftest (id number,name varchar2(20));
Table created.

SQL> begin
  2  for i in 1..100 loop
  3   insert into vftest values (i,'name');
  4   end loop;
  5* end;

PL/SQL procedure successfully completed.
```

查询表 vftest 中的数据。

例子 23-70　查询表 vftest 中的数据

```
SQL> select count(*) from vftest;

  COUNT(*)
----------
       100
```

下面模拟用户错误操作，删除了 ID>80 的数据记录。

例子 23-71　删除 ID>80 的数据记录

```
SQL> delete from vftest
  2  where id>80;

20 rows deleted.

SQL> commit;

Commit complete.
```

此时，我提交了更改，这个更改是无法回滚的，如果没有闪回表特性，我们只能使用备份实现不完全恢复，或者在测试数据库上实现不完全恢复，然后将恢复后的表的数据再导入生产库中去。

下面我们查询表 vftest 的记录。

例子 23-72　查询表 vftest 的记录

```
SQL> select count(*) from vftest;

  COUNT(*)
----------
        80
```

之后，使用闪回表特性。

例子 23-73　使用闪回表

```
SQL>  flashback  table  vftest  to  timestamp  to_date('2011-09-20
18:00:00','yyyy-mm- dd hh24:mi:ss');
flashback table vftest to timestamp to date('2011-09-20 18:00:00','yyyy-mm-dd
hh 24:mi:ss')
                *
ERROR at line 1:
ORA-08189: cannot flashback the table because row movement is not enabled
```

因为没有启动表的行移动特性，无法闪回表，下面启动表 vftest 的行移动特性。

例子 23-74　启动表 vftest 的行移动特性

```
SQL> alter table vftest enable row movement;

Table altered.
```

继续闪回表 vftest 到时间点 2011-09-20 18:00:00，闪回到 5 分钟之前的状态。

例子 23-75　闪回到 5 分钟之前的状态

```
SQL>   flashback  table  vftest  to  timestamp  to date('2011-09-20
18:00:00','yyyy-mm -dd hh24:mi:ss');

Flashback complete.
```

在闪回成功后，需要确定表 vftest 是否闪回到需要的时间点，即我们需要的数据是否恢复了。我们查询表 vftest 的数据。

例子 23-76　查询表 vftest 的数据验证

```
SQL> select count(*) from vftest;

  COUNT(*)
----------
       100
SQL> select max(id) from vftest;

   MAX(ID)
----------
       100
```

从输出看出，我们已经将表 vftest 闪回到需要的状态。在实际工作中，往往需要采用尝试的方法，因为我们已经无法预先知道用户错误操作的具体时间，只能通过大致时间来估算，通过一次次的闪回，并验证表中的数据是否满足需要，一旦满足需要就停止闪回表。

闪回表依然具有操作的局限性，假设当前的时间点为 B 点，需要将表 tt 回到历史上的 A 点，如果在 A 到 B 的这段时间里，对表进行了任何的 DDL 操作，则闪回表操作失败。

23.5　闪回版本查询

所谓版本指的是每次事务所引起的数据行的变化情况，每一次变化就是一个版本。Oracle 提供了闪回版本查询。让我们可以看到数据行的整个变化过程。

上面提到的变化指的是已经提交的事务引起的变化。没有提交的事务引起的变化不会显示，闪回版本查询使用的是 undo 表空间里记录的 undo 数据。闪回版本查询的语法如下所示。

```
select 伪列 1，伪列 2，伪列 3…
from ……
version scn between ……
where 伪列……
```

下面我们解释一下伪列的含义：

- Versions_starttime：事务开始的时间。
- Versions_startscn：事务开始的 SCN 号。
- Versions_endtime：事务结束的时间。
- Versions_endscn：事务结束的 SCN。
- Versions_xid：事务的 ID 号。
- Versions_operation：事务所进行的操作类型，包括插入（显示为 I）、删除（显示为 D）、更新（显示为 U）。

如果我们希望显示数据行的所有的变化，则使用 versions between scn minvalue and maxvalue。

下面我们通过一个例子演示，先创建一个表 vstest，然后执行两次插入操作，一次更新操作并提交，此时一个事务完成。然后再执行一次删除操作并提交，这是第二个事务。整个过程如下所示。

例子 23-77　显示数据行的所有的变化的过程

```
SQL> create table vstest (id number,name varchar2(20));

Table created.

SQL> insert into vstest values(1,'name1');

1 row created.

SQL> insert into vstest values(2,'name2');

1 row created.

SQL> update vstest set name='name22' where id=2;

1 row updated.

SQL> commit;
```

```
Commit complete.
```

完成第一个事务。下面完成第二个事务。

例子 23-78　完成第二个事务

```
SQL> delete from vstest where id=1;

1 row deleted.

SQL> commit;

Commit complete.
```

此时完成第二个事务，接下来，我们通过闪回的版本查询来确认这些行操作的不同版本信息，闪回版本查询如下所示。

例子 23-79　查询闪回版本

```
SQL>select versions starttime "VST",versions startscn "VSS",versions endtime
"VET",
  2      versions endscn   "VES",versions xid   "VXID",versions operation
"VOP",id,name
  3* from vstest versions between scn minvalue and maxvalue

VST            VSS VET                  VES    VXID           V       ID NAME
----------------------------------- - ------------- ---------------------
20-SEP-11 06.25.24 P M  658935        09002F0047010000  D        1  name1
20-SEP-11 06.25.09 P M   658928       0800280019010000  I        2  name22
20-SEP-11  06.25.09  P  M      658928  20-SEP-11  06.25.24  P  M      658935
0800280019010000   I        1  name1
```

23.6　闪回事务查询

闪回事务查询提供的是一个视图，flashback_transaction_query，利用这个视图可以显示是哪些事务引起了数据的变化，并为此提供了撤销事务的 SQL 语句。闪回事务查询利用的是 undo 表空间的 undo 数据。

我们依然使用上一节的两个事务，查询对表 vstest 的两个事务的详细信息。如下所示为闪回事务查询。

例子 23-80　闪回事务查询

```
SQL>  select operation,undo sql from flashback transaction query
  2   where table name='VSTEST';

OPERATION                        UNDO SQL
----------------------------------------------------------------------
UPDATE     update "VFAST"."VSTEST" set "NAME" = 'name2' where ROWID = 'AAA
M2xAAEAAAAGsAAB';
INSERT     delete from "VFAST"."VSTEST" where ROWID = 'AAAM2xAAEAAAAGsAAB' ;
INSERT    delete from "VFAST"."VSTEST" where ROWID = 'AAAM2xAAEAAAAGsAAA';
DELETE    insert into "VFAST"."VSTEST"("ID","NAME") values ('1','name1');
```

上述记录了每个成功提交的事务，这样知道事务的操作，我们通过 UNDO_SQL 可以做相反的操作，将数据库的某个事务闪回。如对应的 DELETE 操作，我们执行 UNDO_SQL 语句来插入数据，从而实现该事务的闪回。接下来我们继续查询事务号和对应的 DML 操作。

例子 23-81　查询事务号和对应的 DML 操作

```
    SQL>  select  xid,operation  from  flashback transaction query  where
table name='VSTEST'
      2* order by xid;

    XID                 OPERATION
    ---------------- --------------------------------
    0800280019010000    UPDATE
    0800280019010000    INSERT
    0800280019010000    INSERT
    09002F0047010000    DELETE
```

从输出知道表 vstest 包含两个事务，一个事务号为 0800280019010000，包括两个插入操作和一个更新操作。一个事务号为 09002F0047010000，包括一个删除操作。这样我们就可以通过 UNDO_SQL 语句将整个事务闪回。

23.7　闪回查询

闪回查询是查询该表过去某个时刻的数据情况，一旦确认某个时刻的数据满足我们的需求以后，可以根据这个时间执行闪回表。和闪回表类似，但是这里强调了逐步闪回，即先查询等确认后再一次闪回到需求时刻。

下面的例子中表 vftest 中开始插入了 10 条记录，然后又添加了 10 条记录，最后删除了 4 条记录。如果我们希望获得开始 10 条记录的状态，并依据该表当时的数据创建一个视图，通过同义词的方式供其他用户使用。

首先，我们需要通过闪回查询获得需要的表数据状态，如例子 23-82 所示。

例子 23-82　通过闪回查询获得需要的表数据状态

```
    SQL>  select count(*)  from  vftest  as  of  timestamp  to_date('2011-09-23
09:25:00','yyyy-mm-dd hh24:mi:ss');

      COUNT(*)
    ----------
            16

    SQL> select  count(*)  from  vftest  as  of  timestamp  to date('2011-09-23
09:15:00','yyyy-mm-dd hh24:mi:ss');

      COUNT(*)
    ----------
            10
```

通过两次闪回查询，我们找到了要闪回到的时间点，下面我们依据该时间点创建一个公有视图，使得其他用户都可以看到。如下所示创建一个视图。

例子 23-83　创建一个视图

```
    SQL> create  view v_vftest_10 as
      2*   select   *   from   vftest   as   of   timestamp   to date('2011-09-23
09:15:00','yyyy-mm-dd hh24:mi:ss')

    View created.
```

创建公有同义词。

例子 23-84　创建公有同义词

```
    SQL> create public synonym v_vftest for v_vftest_10;

    Synonym created.
```

这样我们就可以通过同义词 v_vftest 来查询表 vftest 在过去某个时间点的数据。

23.8　复原点技术

复原点顾名思义就是将数据恢复到该点时的状态，在闪回数据库或者闪回表操作时往往需要一个具体的时间点或者 SCN 等信息 ，说明将闪回到哪里结束，而复原点就是 SCN 的别名，显然复原点更容易记忆。如下所示我们创建一个复原点。

例子 23-85　创建一个复原点

```
    SQL> create restore point rp1;

    还原点已创建。
```

通过数据字典 v$restore_point 查询刚刚创建的复原点 RP1，如例子 23-86 所示。

例子 23-86　查询刚刚创建的复原点 RP1

```
    SQL> select name,scn,storage size,guarantee flashback database
      2  from v$restore point;

    NAME                                SCN      STORAGE SIZE     GUA
    ------------------------------------ --------------------- --------
    RP1                                 619283         0          NO
```

此时成功创建一个复原点，但是该复原点无非是 SCN 的别名，使用起来方便罢了，还是无法确保闪回数据一定成功，因为这依赖于需要的闪回日志数据是否存在，而闪回日志又由快闪恢复区管理。下面我们介绍一种有保证的复原点技术，这种技术使得在快闪恢复区空间保证的情况下总可以闪回到该复原点。如果快闪恢复区空间不足则数据库关闭。

使用有保证的复原点与是否使用闪回日志无关，一旦使用了有保证的复原点，Oracle 会自动保存自复原点之后的闪回日志并不会删除这些日志。下面我们创建一个有保证的复原点，如例子 23-87 所示。

例子 23-87　创建一个有保证的复原点

```
SQL> create restore point grp1 guarantee flashback database;

还原点已创建。
```

在创建完有保证的复原点 grp1 后，我们再次通过数据字典视图 v$restore_point 验证是否创建成功，如例子 23-88 所示。

例子 23-88 通过数据字典视图 v$restore_point 验证是否创建成功。

```
SQL> select name,scn,storage size,guarantee flashback database
  2  from v$restore point;

NAME                                               SCN STORAGE SIZE GUA
--------------------------------------------------------------------
GRP1                                           619784      4096000 YES
RP1                                             619283            0 NO
```

从输出可以看出复原点 GRP1 是有保证的复原点，因为参数 guarantee_flashback_database 为 YES。并且相对于普通复原点 RP1 而言，STORAGE_SIZE 存在参数值，该参数说明为有保证的复原点指定的存储空间。

在不需要复原点时，可以通过 DROP RESTORE POINT 指令删除，删除普通复原点和有保证的复原点的操作相同，如例子 23-89 所示。

例子 23-89　删除普通复原点

```
SQL> drop restore point grp1;

还原点已删除。
```

此时继续查询数据字典以验证删除结果。

例子 23-90　查询数据字典以验证删除结果

```
SQL> select name,scn,storage size,guarantee flashback database
  2  from v$restore_point;

NAME                                               SCN STORAGE_SIZE GUA
---------------------------------------------------------------- ---
RP1                                            619283            0 NO
```

输出显示有保证的复原点 GRP1 已经被删除，当前只有一个普通复原点 RP1。

23.9　本章小结

闪回技术是 Oracle 10g 以后的版本增加的新特性，使用闪回特性可以快速地恢复用户的逻辑错误或者误删除表等操作，闪回删除主要是关注用户误删除表、索引等数据库对象，闪回删除使用回收站作为存储删除的数据库对象的逻辑存储结构，一定要理解闪回删除并不是直接将数据从数据库中删除放入回收站，而只是将数据库对象的定义从数据字典中删除，数据存储在原处，在回收站中记录这个被删除的数据库对象，一旦需要闪回到删除前的状态，使用回收站中记录的对象信息进

行闪回。在删除表时，与表相关联的其他数据库对象如索引、触发器等也都一并删除，在闪回表时，这些相关的数据库对象被自动恢复，但是名称需要进一步修改。

闪回数据库技术是处理用户逻辑错误的快速数据恢复技术，相比传统的数据库恢复会减少恢复时间，传统的数据库恢复时间取决于数据库自身的大小，而闪回数据库技术的恢复时间取决于逻辑错误的数据量大小。使用闪回数据库技术需要将数据库设置为归档模式，并启动快闪恢复区作为闪回日志的存储目录，一般条件下应该尽量设置较大的快闪恢复区，以保证闪回数据库总可以执行成功。

第 24 章

手工管理的备份恢复

Oracle 数据库提供了完备的数据库备份恢复方法以及工具，手工管理的备份恢复需要 DBA 参与备份恢复的全过程，并且针对不同的备份策略，数据库故障原因执行不同的恢复过程。在备份过程中需要 DBA 启动不同粒度的备份模式，使用操作系统工具备份数据库文件，关闭备份模式，在恢复阶段需要根据当时的故障造成的数据库状态执行对应的策略，如将丢失的数据文件 OFFLINE，保证数据库可以打开，使用操作系统工具复原数据文件，以及 RECOVER 数据库等操作。显然，人工管理的备份恢复全程需要 DBA 的参与，对 DBA 的要求较高，因此 DBA 需要充分理解热备冷备的原理以及归档模式的含义，并且熟悉 Oracle 数据库的启动过程对 SCN 的处理等。本章的学习将使得读者对人工管理的备份恢复有一个充分的理解和具体的把握。

24.1　备份恢复的概念

数据库备份是 DBA 一项十分重要的任务，使用备份的数据库文件可以在数据库出现人为或设备故障时迅速的恢复数据，保证数据库系统对外提供持续，一致的数据库服务。备份是数据库的一个副本，具体内容包括数据文件、控制文件等（也可以是逻辑备份），通过备份 DBA 可以有效防止不可预测的数据丢失或应用程序错误造成的数据丢失，通过备份有效还原数据。

本节我们引入几个关键概念，理解这些基本概念对于实施备份恢复措施，选择合理的备份恢复方案十分重要。

24.1.1　物理备份

物理备份是指将数据库文件（如数据文件、控制文件以及重做日志文件）复制到指定目录作为数据文件备份的方式，采用物理备份时无论数据库文件中是否有数据，会复制整个数据文件，显然物理备份会增加备份的存储空间，需要 DBA 事先查看数据库文件的大小，使用合理的存储空间。物理备份有两种实现方式。

使用操作系统工具 CP 和 DD 来备份和管理数据文件。

使用 Oracle 实用工具 RMAN（Recovery Manager）来备份和管理数据文件，使用 RMAN 时既

可以使用命令行方式（RMAN 的命令行），也可以使用 GC 或 EM 图像化工具实现。

24.1.2 逻辑备份

逻辑备份是指使用 Oracle 提供的数据迁移工具如 EXPDP、EXP 等，导出数据库对象的逻辑结构以及数据，此时会先导出数据库对象的结构，如表的定义或者所标的结构、索引的定义等，然后导出相应的数据库对象中的数据。逻辑备份是 DBA 可以采用的备份方式之一，但是使用逻辑备份时，如果是恢复整个数据库往往需要很长的时间，而如果此时使用物理备份相对快得多，因为物理备份主要是复制数据文件的过程，而逻辑备份的恢复需要 Oracle 数据库创建所有的数据库对象，并插入相应的数据。逻辑备份不能替代物理备份。

24.1.3 冷备份与热备份

冷与热是对数据库运行状态的十分形象的表示，冷是指在数据库关闭的情况下的备份，如果数据库是正常关闭的如 shutdown immediate，此时冷备份是一致的数据库备份。热备份是指在数据库打开的状态下的备份，此时用户可以继续访问数据库，执行 DML 操作。但是要求此时的数据库必须运行在归档模式，归档模式下可以对整个数据库、单独的表以及数据文件进行备份。如果数据库运行在非归档模式，则会提示如下错误。

```
SQL> alter tablespace users begin backup;
alter tablespace users begin backup
*
ERROR at line 1:
ORA-01120: cannot start online backup; media recovery not enabled
```

24.1.4 数据库恢复

1. 实例恢复

实例是 Oracle 非常重要的概念，实例包括内存结构和后台进程。实例故障是指实例突然失败造成的数据库故障，如使用 shutdown abort 关闭数据库，此时再启动数据库时就会实施实例恢复。实例恢复使用当前的数据文件以及当前的重做日志文件实现数据恢复，实例恢复由 Oracle 数据库自动完成，不需要用户的干预。

实例恢复由两个步骤完成，一个是前滚（roll forward），一个是回滚（rollback）。下面介绍这两个步骤要做什么。

前滚（roll forward）将重做日志文件中记录的用户提交和未提交的事务涉及的数据写入联机数据文件。实例恢复前滚示意图如图 24-1 所示。

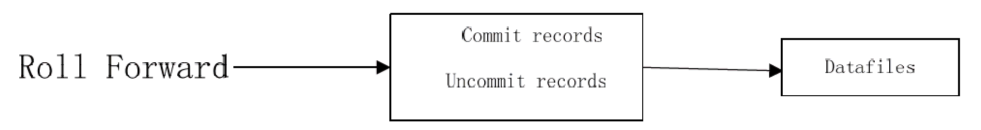


图 24-1 实例恢复前滚示意图

回滚（rollback）将未提交事务涉及的修改通过 UNDO 中的记录回退回去，即如果原先未提交

事务涉及了 DELETE 操作，回滚就是使用 UNDO 记录将删除的数据 INSERT 操作写回当前的数据文件，以保证事务的原子性，如图 24-2 所示。

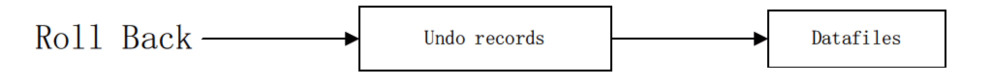


图 24-2　实例恢复回滚示意图

实例恢复将读取重做日志，然后进行前滚和回滚操作。读者或许会问，如果重做日志文件很大，恢复的时间就会很长吗？其实这个问题确实存在，但是 Oracle 已经预感到这个问题，在 Oracle 10g 中使用参数 fast_start_mttr_target 使得 Oracle 定时将脏数据写入数据文件，来调整检查点，这样检查点之前的提交事务部需要前滚操作，只有检查点之后的提交事务才需要前滚，如果这个时间间隔很小，显然实例恢复的时间就减少，但是频繁的 checkpoint 同样会影响数据库性能，这就需要根据数据库的实际做出合理的设计。

2. 介质恢复

介质恢复是由于磁盘损坏或者数据文件损坏而需要的数据库恢复方式，介质恢复不是 Oracle 数据库自动完成的，需要用户干预来启动整个介质恢复过程。介质恢复必须具有备份的数据文件，如果需要完全恢复还需要自备份以来的归档日志文件，当前的日志文件。

介质恢复需要两个步骤，即复原（Restore）和恢复（Recover）。

复原是指将备份的数据文件复制到相应目录，如果不是该文件以前的目录则需要使用 alter database rename datafile 指令来告诉数据库新的数据文件的位置。

恢复是指使用归档重做日志以及当前的日志文件恢复数据库到最新的状态，恢复包含两个步骤，其实，这两个步骤就是实例恢复的步骤：前滚和回滚。其中前滚将利用重做日志（包括归档日志和当前日志）来应用提交和未提交的事务。这个过程显然包含了一些不一致的数据，即未提交的数据，对应的事务需要回滚，回滚是恢复的第二个步骤，通过 UNDO 记录中的原始数据来恢复未提交事务更改的数据，保证事务的原子性。

3. 完全和不完全恢复

完全恢复是指当数据库发生故障后，通过备份将数据库恢复到最新的状态，不丢失任何数据，如果数据库有数据文件的备份，并且有备份以来全部的归档日志文件和当前的日志文件，就不会丢失数据，可以不同力度实现完全恢复，如数据库级、表空间级和数据文件级。

不完全恢复不能把数据库恢复到最新状态，会有数据丢失，可以将数据库恢复到指定的时间点，通常不完全恢复往往是由于丢失归档日志或当前日志造成的，只能在数据库级实现不完全恢复，在表空间级以及数据文件级不能实现不完全恢复。

24.2　非归档模式下的冷备与恢复

冷备份是关闭数据库后的数据库备份，此时的备份一定是一致的数据库备份，因为数据库安全关闭。冷备是最简单也是最可靠的备份方法。简单是因为只需要复制需要的数据库文件，这些文件包括数据文件、控制文件以重做日志文件。可靠是因为在数据库安全关闭的条件下备份数据库文

件，此时是一致的数据库备份。

冷备的缺点同样很明显，冷备要备份全部数据库文件，如果超大型数据库则需要较大的存储介质，如果数据变化很小，则为了备份这些变化的数据同样需要备份整个数据库，显然这样的备份效率不高。

24.2.1 冷备的步骤

冷备最明显的特点就是安全关闭数据库，然后通过操作系统指令复制数据库文件，这里列出冷备的具体步骤，然后通过一个表空间的冷备说明如何具体实施冷备。下面是非归档模式下的全库冷备的步骤。

01 查看数据库文件的存储目录，如控制文件、数据文件和重做日志文件，此时需要使用数据字典 dba_data_files、v$controlfile、v$logfile 明确需要备份的数据库文件。

例子 24-1 查看数据库文件的存储目录

```
SQL> select file_name from dba_data_files

FILE_NAME
--------------------------------------------------------
/u01/app/oracle/oradata/TEST/users01.dbf
/u01/app/oracle/oradata/TEST/sysaux01.dbf
/u01/app/oracle/oradata/TEST/undotbs01.dbf
/u01/app/oracle/oradata/TEST/system01.dbf
/u01/app/oracle/oradata/TEST/example01.dbf
```

查看控制文件。

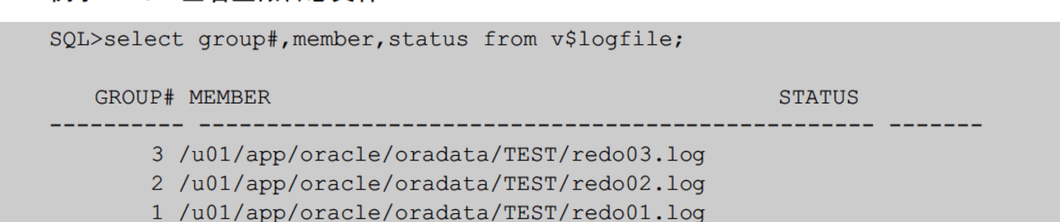
例子 24-2 查看控制文件

```
SQL> select name from v$controlfile

NAME
--------------------------------------------------------
/u01/app/oracle/oradata/TEST/control01.ctl
/u01/app/oracle/oradata/TEST/control02.ctl
/u01/app/oracle/oradata/TEST/control03.ctl
```

查看重做日志文件。

例子 24-3 查看重做日志文件



```
SQL>select group#,member,status from v$logfile;

    GROUP# MEMBER                                             STATUS
---------- -------------------------------------------------- -------
         3 /u01/app/oracle/oradata/TEST/redo03.log
         2 /u01/app/oracle/oradata/TEST/redo02.log
         1 /u01/app/oracle/oradata/TEST/redo01.log
```



在生产数据库中，重做日志文件以及控制文件应该实现冗余配置，上例中的控制文件和重做日志文件以及数据文件我们都采用默认的安装设置，都在一个目录下，目的是为了演示方便而已。

02 安全关闭数据库。

例子 24-4　安全关闭数据库

```
SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
```

03 复制文件到指定的备份目录。

首先创建备份目录。

例子 24-5　创建备份目录

```
[oracle@ocm1 ~]$ mkdir /u01/backup
[oracle@ocm1 ~]$ mkdir /u01/backup/datafile
[oracle@ocm1 ~]$ mkdir /u01/backup/controlfile
[oracle@ocm1 ~]$ mkdir /u01/backup/redofile
```

复制数据文件。

```
[oracle@ocm1 ~]$ cp /u01/app/oracle/oradata/TEST/*.dbf /u01/backup/datafile
```

复制控制文件。

```
[oracle@ocm1 ~]$ cp /u01/app/oracle/oradata/TEST /*..ctl
/u01/backup/controlfile
```

复制重做日志文件。

```
[oracle@ocm1 ~]$ cp /u01/app/oracle/oradata/TEST /*.log /u01/backup/redofile
```

04 打开数据库。

例子 24-6　打开数据库

```
SQL> startup
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                  1218992 bytes
Variable Size              92276304 bytes
Database Buffers          188743680 bytes
Redo Buffers                2973696 bytes
Database mounted.
Database opened.
```



非归档模式备份的数据库会造成从备份开始到故障发生之间的所有用户提交的数据丢失，所以使用备份只能恢复到冷备开始时，而且冷备的恢复需要关闭数据库，这就要求冷备的频率较高，以减少数据库崩溃造成的数据丢失。

24.2.2 冷备下的恢复

在演示如何恢复之前，我们先模拟一个故障，我们首先创建一个表，然后插入数据，最后删除数据，模拟用户的错误操作，看使用冷备份如何实现误操作的恢复。

创建一个新表，并插入数据。

例子 24-7 创建一个新表，并插入数据

```
SQL> create table t1(id number,name varchar2(20));

Table created.

SQL> insert into t1 values(1,'name1');

1 row created.

SQL> commit;

Commit complete.
```

在上面的操作中，我们创建了表 t1 并插入了部分数据。接下来模拟数据损坏的故障，我们删除数据文件 users01.dbf。

例子 24-8 删除文件模拟故障

```
[oracle@ocm1 backup]$ cd /u01/app/oracle/oradata/TEST
[oracle@ocm1 TEST]$ ls
control01.ctl  example01.dbf  redo03.log    temp01.dbf
control02.ctl  redo01.log     sysaux01.dbf  undotbs01.dbf
control03.ctl  redo02.log     system01.dbf  users01.dbf
[oracle@ocm1 TEST]$ rm -rf users01.dbf                   //删除数据文件
[oracle@ocm1 TEST]$ ls
control01.ctl  example01.dbf  redo03.log    temp01.dbf
control02.ctl  redo01.log     sysaux01.dbf  undotbs01.dbf
control03.ctl  redo02.log     system01.dbf
```

接下来重启数据库。

例子 24-9 重启数据库

```
SQL> startup
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                  1218992 bytes
Variable Size              92276304 bytes
```

```
Database Buffers        188743680 bytes
Redo Buffers             2973696 bytes
Database mounted.
ORA-01157: cannot identify/lock data file 4 - see DBWR trace file
ORA-01110: data file 4: '/u01/app/oracle/oradata/TEST/users01.dbf'
```

提示数据文件丢失。下面恢复数据库，首先关闭数据库。

例子 24-10　关闭数据库

```
SQL> shutdown immediate;
ORA-01109: database not open


Database dismounted.
ORACLE instance shut down.
```

复制回目录、数据文件、控制文件。

例子 24-11　复制回相关文件

```
[oracle@ocm1 backup]$ cp /u01/backup/datafile/*.* /u01/app/oracle/oradata/TEST
[oracle@ocm1 ~]$ cp /u01/backup/controlfile/*.* /u01/app/oracle/oradata/TEST
[oracle@ocm1 ~]$ cp /u01/backup/redofile/*.* /u01/app/oracle/oradata/TEST
```

打开数据库。

例子 24-12　打开数据库

```
SQL> startup
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                 1218992 bytes
Variable Size             92276304 bytes
Database Buffers         188743680 bytes
Redo Buffers               2973696 bytes
Database mounted.
Database opened.
```

打开数据库后，查看数据文件是否存在。

例子 24-13　查看数据文件是否存在

```
SQL> select file name from dba data files;

FILE_NAME
-------------------------------------------------------
/u01/app/oracle/oradata/TEST/users01.dbf
/u01/app/oracle/oradata/TEST/sysaux01.dbf
/u01/app/oracle/oradata/TEST/undotbs01.dbf
/u01/app/oracle/oradata/TEST/system01.dbf
/u01/app/oracle/oradata/TEST/example01.dbf
```

接下来，通过 DESC 指令查看表 t1 是否存在。

例子 24-14　通过 DESC 指令查看表 t1 是否存在

```
SQL> desc t1;
ERROR:
ORA-04043: object t1 does not exist
```

显然此时表 t1 已经无法恢复了，表定义以及表数据丢失，如果此时只恢复数据文件 users01.dbf，则表的定义应该存在，但是表中的数据丢失无法恢复。整个冷备的恢复过程最耗费时间的就是复制数据文件。

数据库的故障是数据文件丢失，此时我们使用冷备完成了恢复，如果用户错误地删除了一个表（如 t2 表），该表在上次冷备时存在且有部分数据，则可以通过冷备恢复 2 表以及部分数据，但是从上次备份以来的所有数据将丢失。如果表 t2 中只有少量数据需要恢复，则使用冷备仍然需要恢复整个数据库，Oracle 为了解决这个问题引入了闪回技术，使用闪回技术将方便恢复用户的逻辑操作错误，闪回将在下面的章节介绍。

24.2.3　缺少重做日志文件的恢复方法

备份了控制文件和数据文件，但是没有备份重做日志文件，此时需要启动到 MOUNT 状态，使用 recover database until cancel using backup controlfile，然后通过 alter database open resetlogs 来打开数据库。具体步骤和详细解释如下所示。

01 将备份的数据文件和重做日志文件复制到相应目录下。

```
[oracle@ocm1 backup]$ cp *.ctl /u01/app/oracle/oradata/TEST
[oracle@ocm1 backup]$ cp *.dbf /u01/app/oracle/oradata/TEST
```

在模拟这个故障时，将数据库文件全部删除了，也就是重做日志文件根本没有，然后我们启动数据库观察有什么错误提示。

例子 24-15　启动数据库观察是否有错误提示

```
SQL> startup
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                 1218992 bytes
Variable Size             92276304 bytes
Database Buffers         188743680 bytes
Redo Buffers               2973696 bytes
Database mounted.
ORA-00313: open failed for members of log group 1 of thread 1
ORA-00312: online log 1 thread 1: '/u01/app/oracle/oradata/TEST/redo01.log'
```

显然，提示没有日志文件，说明在控制文件中第一个标识的数据库文件就是该日志文件。

02 使用备份的控制文件恢复数据库。

例子 24-16　使用备份的控制文件恢复数据库

```
SQL> recover database until cancel using backup controlfile;
ORA-00279: change 483797 generated at 09/06/2011 20:51:42 needed for thread
```

```
1
    ORA-00289: suggestion :
    /u01/app/oracle/flash_recovery_area/TEST/archivelog/2011_09_07/o1_mf_1_2_%
u .arcORA-00280: change 483797 for thread 1 is in sequence #2


    Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
    cancel
    Media recovery cancelled.
```

在 Oracle 要求指定日志时，选择了 CANCEL，该参数的含义是让 Oracle 到归档日志中寻找需要的日志数据，满足条件则自动停止。

03 打开数据库。

例子 24-17 打开数据库

```
    SQL> alter database open;
    alter database open
    *
    ERROR at line 1:
    ORA-01589: must use RESETLOGS or NORESETLOGS option for database open


    SQL> alter database open resetlogs;

    Database altered.
```

使用备份的控制文件必须使用 resetlogs 参数打开数据库，此时的数据库是一个新的“化身”。下面我们查看一下当前重做日志的信息。

例子 24-18 查看一下当前重做日志的信息

```
    SQL> select group#,status,member from v$logfile

       GROUP# STATUS  MEMBER
    ---------- ------- --------------------------------------------------
             3        /u01/app/oracle/oradata/TEST/redo03.log
             2        /u01/app/oracle/oradata/TEST/redo02.log
             1        /u01/app/oracle/oradata/TEST/redo01.log
```

通过输出知道，Oracle 通过控制文件中记录的重做日志文件的信息重建了重做日志组以及重做日志成员。重做日志的状态信息如下。

例子 24-19 重做日志的状态信息

```
    SQL> select group#,sequence#,bytes,status from v$log;

       GROUP#  SEQUENCE#      BYTES STATUS
    ---------- ---------- ---------- ----------------
             1          1   52428800 INACTIVE
             2          2   52428800 CURRENT
             3          0   52428800 UNUSED
```

至此，整个数据库得以恢复，但是自备份以来的所有提交数据丢失。

24.3 归档模式与非归档模式

归档模式是 Oracle 提供的一种数据库完全恢复的重要措施，Oracle 的重做日志文件是循环使用的。在切换日志时，如果启动了归档模式，后台归档进程 ARCH 就会启动，当前重做日志中的记录就会存储到归档目录下，形成归档文件。归档数据记录了 Oracle 数据库自启动归档以来所有的数据变化信息，使用归档和备份的数据库文件可以实现数据库的完全恢复，当然还需要其他文件如当前的重做日志文件以及控制文件。

下面我们详细介绍如何启动归档模式，以及归档目录需要注意的问题。下面演示如何启动数据库到归档模式。

24.3.1 设置数据库的归档模式

归档可以自动进行，也可以手动执行。所谓自动进行就是在事件触发的条件下，Oracle 数据库自动启动归档进程完成重做日志数据的归档。只要有足够的磁盘空间归档过程就可以持续下去，一旦启动了归档模式，就可以完成很多操作，当然最重要的是可以实现数据库的完全恢复，对数据库实现联机热备并且可以实现不完全恢复。

使用 SYS 用户登录数据库，将数据库启动到 MOUNT 状态，如例子 24-20 所示。

例子 24-20　将数据库启动到 MOUNT 状态

```
[oracle@ocm1 ~]$ sqlplus sys/oracle as sysdba

SQL*Plus: Release 11.2.0.1.0 - Production on Sat Aug 27 07:06:38 2011

Copyright (c) 1982, 2009, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, Oracle Label Security, OLAP and Data Mining options
SQL> startup mount;
ORACLE instance started.

Total System Global Area  314572800 bytes
Fixed Size                  1219184 bytes
Variable Size              88081808 bytes
Database Buffers          222298112 bytes
Redo Buffers                2973696 bytes
Database mounted.
```

下面，打开数据库并查看当前的归档模式。

例子 24-21　打开数据库并查看当前的归档模式

```
SQL> alter database open;
```

```
Database altered.

SQL> archive log list;
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination           /u01/app/oracle/oracle/product/10.2.0/db_1/dbs/arch
Oldest online log sequence     62
Next log sequence to archive   66
Current log sequence           66
```

从输出看出 Database log mode 显示为 Archive Mode 归档模式。在 Oracle 11g 以及更高的版本中，一旦启动归档模式，后台归档进程就自动启动，这样归档行为也就自动进行，即一旦发生日志组切换就会触发归档进程。如下我们查看 PROD 库的归档进程信息。

例子 24-22　查看 PROD 库的归档进程信息

```
[oracle@ocm1 ~]$ ps -ef |grep arc
oracle   20331     1  0 07:16 ?        00:00:00 ora_arc0_PROD
oracle   20333     1  0 07:16 ?        00:00:00 ora_arc1_PROD
oracle   20335     1  0 07:16 ?        00:00:00 ora_arc2_PROD
oracle   20442 20000  0 07:19 pts/2    00:00:00 grep arc
```

从输出可以看出数据库 PROD 启动了归档进程。相反，一旦关闭归档模式，则后台归档进程会自动关闭。

在 Oracle 10g 之前的归档模式下，归档进程需要手工启动，而且在数据库重启之后需要重新启动。如在 Oracle 9i 中的情况，需要使用如下指令手工归档重做当前的重做日志。

例子 24-23　手工归档重做当前的重做日志。

```
SQL> alter system archive log current;

System altered.
```



无论在手工归档模式还是在自动归档模式下都可以使用上述指令完成日志的归档工作。

在 Oracle 10g 之前同样可以设置数据库的自动归档模式，此时有两种方式，一种是在数据库运行期间使用如下指令实现。

例子 24-24　使用指令设置数据库的自动归档模式

```
SQL> alter system archive log start;

System altered.
```

但是，这种方式仍然存在潜在危险，就是一旦数据库关闭，重启数据库后归档进程不会自动启动。所以，为保险起见最好使用修改初始化参数的方法。如在数据库运行时使用如下指令。

例子 24-25　使用修改初始化参数的方法

```
SQL> alter system  set log archive start=true scope=spfile;

System altered.
```

因为该参数是静态参数，所以需要重启数据库使其生效，该参数的修改使得每次启动数据库时归档进程会自动启动。



在 Oracle 10g 以上版本中，不需要修改这个参数，如果无意修改了这个参数，Oracle 不会报错，但是每次启动数据库时会提示 ORA-32004: obsolete and/or deprecated parameter(s) specified。

24.3.2　设置归档进程相关参数

我们已经知道归档进程的作用是将重做日志文件备份到指定的归档目录，这里重温一下这个过程将有助于理解归档进程相关参数的设置，如图 24-3 所示。

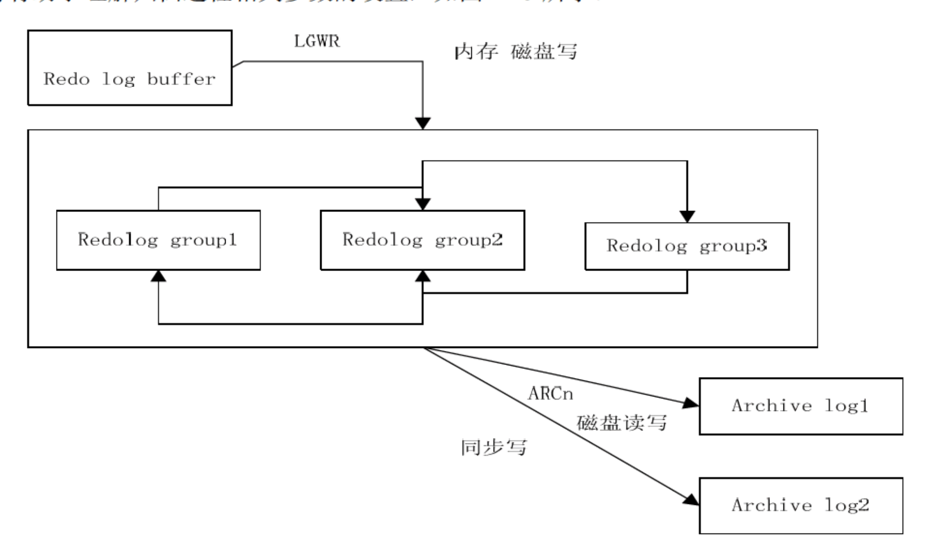


图 24-3　重做日志以及归档日志的工作过程图

在图 24-3 中，首先 LGWR 进程会将 REDO LOG BUFFER 中的重做数据写入 REDO LOG ，此时的 REDO LOG 分组，每当一个 REDO LOG GROUP 满时，或者发出 SWITCH LOGFILE 指令时都会触发日志组的切换，当发生日志组切换时，ARC 进程会将当前的重做日志中数据写入归档目录，我们需要理解一个问题就是 LGWR 进程是将内存中的数据写入重做日志文件，是内存读磁盘写，而 ARC 进程是将重做日志文件写入归档文件，是磁盘读磁盘写，显然 LGWR 进程的读写效率或者读写速度比 ARC 进程要快，而频繁发生 DML 操作的数据库中，可能会发生由于归档慢而重做日志写入速度快造成的数据库被 HANG 住了，此时数据库什么也不做就是等待 ARC 进程将当前的重做日志数据写入归档文件。为了匹配二者的速度问题可以考虑修改归档进程参数 log_archive_max_processes，它的作用就是提高归档进程的数量，启动更多的归档进程完成写归档

的过程。

默认情况下，该参数的值是 2。由于参数 log_archive_max_processes 是动态参数，我们可以直接修改。

在修改之前先查询当前的该参数值，如例子 24-26 所示。

例子 24-26　查询当前的 log_archive_max_processes 参数值

```
SQL> show parameter log archive max processes;

NAME                        TYPE                        VALUE
------------------------------------------------- -----------------
log_archive_max_processes       integer                          2
```

下面，修改最大归档进程数为 3。

例子 24-27　修改最大归档进程数为 3

```
SQL> alter system set log archive max processes=3;

System altered.
```

提示已经更改完毕，此时我们在内存和 SPFILE 中同时修改了该参数，下面通过 SHOW PARAMETER 指令查看修改结果。

例子 24-28　通过 SHOW PARAMETER 指令查看修改结果

```
SQL> show parameter log_archive_max_processes;

NAME                        TYPE             VALUE
---------------------------------------------- -------------------
log_archive_max_processes       integer             3
```

显然此时的最大归档进程数已经是 3 个了，然后我们通过操作系统验证是否启动了更多的归档进程。

例子 24-29　验证是否启动了更多的归档进程

```
[oracle@ocm1 ~]$ ps -ef|grep arc
oracle    7760     1  0 06:22 ?       00:00:00 ora arc0 PROD
oracle    7762     1  0 06:22 ?       00:00:00 ora_arc1_PROD
oracle    8704     1  0 06:40 ?       00:00:00 ora_arc2_PROD
oracle    8784  8764  0 06:41 pts/2    00:00:00 grep arc
```

从输出看出此时有 3 个后台归档进程：ora_arc0_PROD、ora_arc1_PROD、ora_arc2_PROD。从中也可以看出 Oracle 对归档进程的命名即 ORA_ARCn_实例名。

24.3.3　管理归档文件和归档目录

在上一节，通过修改参数 log_archive_max_processes 解决了 LGWR 写入速度和 ARC 写入速度的不匹配问题，接下来讨论归档日志的安全问题。我们知道如果只有一份归档日志的话，一旦该归档日志损坏或者所在的磁盘故障，都会造成数据库无法完全恢复。此时就需要考虑如何通过冗余来

解决归档日志文件不安全的问题。

在 Oracle 中可以设置最多 10 个归档目录，也就是可以设置 10 个不同的归档地址，通过冗余来保障归档日志的安全。我们先看看相关的参数，如下所示。

例子 24-30　查看归档日志文件相关的参数

```
SQL> show parameter log_archive_dest;

NAME                                 TYPE                           VALUE
------------------------------------ ------------------------------
log_archive_dest                      string
log_archive_dest_1                    string
log_archive_dest_10                   string
log_archive_dest_2                    string
log_archive_dest_3                    string
log_archive_dest_4                    string
log_archive_dest_5                    string
log_archive_dest_6                    string
log_archive_dest_7                    string
log_archive_dest_8                    string
log_archive_dest_9                    string

NAME                                 TYPE                           VALUE
------------------------------------ ------------------------------
log_archive_dest_state_1              string                         enable
log_archive_dest_state_10             string                         enable
log_archive_dest_state_2              string                         enable
log_archive_dest_state_3              string                         enable
log_archive_dest_state_4              string                         enable
log_archive_dest_state_5              string                         enable
log_archive_dest_state_6              string                         enable
log_archive_dest_state_7              string                         enable
log_archive_dest_state_8              string                         enable
log_archive_dest_state_9              string                         enable
SQL>
```

参数 log_archive_dest_n 表示归档目录，可以设置最多 10 个，该参数的值为空说明没有设置具体的归档目录，而 log_archive_dest_state_n 表示归档目录的状态，enable 说明这个参数的目录有效，Oracle 可以使用。这个参数的具体使用我们在最后会讨论。

为了设置多个归档目录，我们创建了 3 个目录/u01/arc1、/u01/arc2、/u01/arc3 来模拟 3 个磁盘，在 3 个不同的物理磁盘上创建归档目录。

```
[oracle@ocm1 ~]$ mkdir /u01/arc1 -p
[oracle@ocm1 ~]$ mkdir /u01/arc2 -p
[oracle@ocm1 ~]$ mkdir /u01/arc3 -p
[oracle@ocm1 ~]$ ls /u01/
app  arc1  arc2  arc3
```

下面使用 alter system set 指令设置 3 个归档目录。

例子 24-31 使用 alter system set 指令设置 3 个归档目录

```
SQL> alter system set log archive dest 1='location=/u01/arc1 mandatory';

System altered.
SQL> alter system set log archive dest 2='location=/u01/arc2 optional';

System altered.
SQL> alter system set log_archive_dest_3='location=/u01/arc3 ';

System altered.
```



参数 optional 是默认的，如果我们没有指定这里的参数为 mandatory 或者 optional，默认就是 optional。

参数 mandatory 要求该归档目录必须写成功，即 Oracle 要求日志数据必须归档到该目录成功后才可以切换，而 optional 是可选的意思，即该目录没有归档成功也可以切换日志，如果该目录损坏某种程度上不影响日志归档。

为了安全起见，我们查看设置结果。

例子 24-32 查看设置结果

```
SQL> show parameter log archive

NAME                          TYPE        VALUE
---------------------------------------------------------------------------
log archive config            string
log archive dest              string
log archive dest 1            string      location=/u01/arc1 mandatory
log_archive_dest_2            string      location=/u01/arc2 optional
log_archive_dest_3            string      location=/u01/arc3
…….
log archive dest state 1      string      enable
log archive dest state 2      string      enable
log_archive_dest_state_3      string      enable
……
```

输出显示 3 个归档目录都设置成功，而且状态都为可用（enable）状态。下面我们执行一次日志切换启动归档，查看这些目录下是否有归档文件。

例子 24-33 查看这些目录下是否存在归档文件

```
SQL> alter system switch logfile;

System altered.
```

然后查看归档目录下的是否有归档文件。

例子 24-34 查看归档目录下的是否存在归档文件

```
 [oracle@ocm1 ~]$ ls /u01/arc1
1_68_743979786.dbf
```

```
[oracle@ocm1 ~]$ ls /u01/arc2
1_68_743979786.dbf
[oracle@ocm1 ~]$ ls /u01/arc3
1_68_743979786.dbf
```

可以看出，此时每个目录下都有一个归档文件，而且名字相同，符合我们的要求。既然 Oracle 提供了两个参数 mandatory 和 optional，且以上三个归档目录一个是 mandatory 两个为 optional，那么如何强制实现多个归档目录必须成功呢，就是要求必须有多个归档目录归档。Oracle 提供了参数 log_archive_min_succeed_dest 来保证最小的成功归档地址数量，如设置 log_archive_min_succeed_dest=2 则表示至少有两个归档地址归档成功，当前的日志组才可以切换。下面设置该参数。

例子 24-35　设置 log_archive_min_succeed_dest 参数

```
SQL> alter system set log_archive_min_succeed_dest=2;

System altered.

SQL> show parameter log_archive_min_succeed_Dest;

NAME                                 TYPE                   VALUE
------------------------------------ ------------------------------
log_archive_min_succeed_dest         integer                2
```

上面，我们设置了 log_archive_min_succeed_dest 为 2，而我们设置了 3 个归档目录，只有一个是 mandatory，那么此时 Oracle 将从 optional 中选择一个作为强制归档成功的目录。

如果设置的该参数 log_archive_min_succeed_dest 值为 4，则报错提示归档目录数少于设置数。

例子 24-36　设置参数 log_archive_min_succeed_dest 值为 4 的错误

```
SQL> alter system set log_archive_min_succeed_dest=4;
alter system set log_archive_min_succeed_dest=4
*
ERROR at line 1:
ORA-02097: parameter cannot be modified because specified value is invalid
ORA-16020: less destinations available than specified by
LOG_ARCHIVE_MIN_SUCCEED_DEST
```

24.4　手工热备数据库的步骤

热备是数据库运行过程中的数据库备份，手工管理的热备要求使用操作系统工具将数据库文件备份到指定的目录。热备可以备份某个表空间的所有数据文件，也可以备份某个表空间的一个数据文件，在备份过程中表空间依然可用，DML 操作依然支持。

在热备过程中数据文件头的 SCN 被锁定，此时不会有任何数据再写入数据文件，那么为什么还支持 DML 操作呢？读者如果熟悉 Oracle 重做日志的原理就会明白，此时数据的变化都写入了重做日志文件，当表空间或数据文件结束备份模式时，会触发恢复过程，将涉及该表空间的所有变化数据写入该表空间。提升数据文件头部的 SCN。使得控制文件和数据文件中记录的 SCN 一致。

热备的前提是数据库必须出于归档模式，此时一般只需要备份数据文件，而对于控制文件，重做日志文件以及归档日志文件可以不考虑备份，因为这些文件我们可以通过前期的冗余设计来实现，也就是连接热备时，我们只需要考虑备份数据文件即可。

热备的好处很多，最大的好处就是不必像冷备那样关闭数据库，这对于如银行、证券、移动、电信这样要求不停机运行的数据库环境就极为重要，此时应用系统的正常运行，用户不会感到数据库的备份行为，备份行为对用户透明。热备的缺点是归档模式下数据库的管理复杂，系统开销增大，对 DBA 提出更高的要求。

我们在上一节已经经过如何设置归档模式，以及如何管理归档目录。下面是手工热备的具体步骤，已备份表空间 USERS 为例。

- 将表空间至于备份模式，使用 alter tablespace users begin backup 指令。
- 使用操作系统工具将表空间对应的数据文件备份到指定目录。
- 结束备份模式，使用 alter tablespace users end backup 指令。
- 归档当前日志，备份归档日志。

下面我们给出具体的操作步骤，以及热备过程中的注意事项。将表空间置于备份模式，首先查看 USERS 表空间对应的数据文件。

例子 24-37 查看 USERS 表空间对应的数据文件

```
SQL> col file name for a47
SQL> col tablespace name for a10
SQL> select file id,file name,tablespace name from dba data files
  FILE_ID FILE_NAME                                 TABLESPACE
--------------- ------------------------------ ------------------------
        1 /u01/app/oracle/oradata/PROD/system01.dbf      SYSTEM
        2 /u01/app/oracle/oradata/PROD/undotbs01.dbf     UNDOTBS
        3 /u01/app/oracle/oradata/PROD/sysaux01.dbf      SYSAUX
        4 /u01/app/oracle/oradata/PROD/users01.dbf       USERS
        7 /u01/app/oracle/oradata/PROD/users02.dbf       USERS
        8 /u01/app/oracle/oradata/PROD/audit01.dbf       AUDIT TBS

6 rows selected.
```

我们看到表空间 USERS 有两个数据文件，文件 ID 分别为 4 和 7，备份时就需要复制这两个数据文件到指定的备份目录。

接着将表空间置于备份模式。

例子 24-38 将表空间置于备份模式

```
SQL> alter tablespace users begin backup;

Tablespace altered.
```

我们通过数据字典 v$backup 来查看数据文件是否处于备份状态。

例子 24-39 查看数据文件的是否处于备份状态

```
SQL> select * from v$backup;
```

```
    FILE# STATUS                CHANGE# TIME
---------- ------------------ ---------- ---------
         1 NOT ACTIVE                 0
         2 NOT ACTIVE                 0
         3 NOT ACTIVE                 0
         4 ACTIVE               1188161 31-AUG-11
         7 ACTIVE               1188161 31-AUG-11
         8 NOT ACTIVE                 0

6 rows selected.
```

USERS 表空间对应两个数据文件，ID 分别为 4 和 7。此时 STATUS 为 ACTIVE，说明该数据文件处于备份模式。此时数据文件的文件 SCN 为 1188161，该 SCN 在备份期间不会变化。

备份数据文件到指定的目录。

例子 24-40　备份数据文件到之前的目录

```
[oracle@ocm1 ~]$ cp /u01/app/oracle/oradata/PROD/user*.* /u01/backup
[oracle@ocm1 ~]$ ls /u01/backup
users01.dbf  users02.dbf
```

通过操作系统工具备份了数据文件，同时使用 ls 指令查看了目录/u01/backup 下确实存在表空间 USERS 的两个数据文件。

结束表空间的备份模式。

例子 24-41　结束表空间的备份模式

```
SQL> alter tablespace users end backup;

Tablespace altered.
```

结束表空间的备份状态后，表空间对应的所有数据文件头解锁，数据库对该表空间的操作恢复正常。

然后，通过数据字典 v$backup 查看数据文件是否结束了备份模式。

例子 24-42　查看数据文件是否结束了备份模式

```
SQL> select * from v$backup;

    FILE# STATUS                CHANGE# TIME
---------- ------------------ ---------- ---------
         1 NOT ACTIVE                 0
         2 NOT ACTIVE                 0
         3 NOT ACTIVE                 0
         4 NOT ACTIVE           1188161 31-AUG-11
         7 NOT ACTIVE           1188161 31-AUG-11
         8 NOT ACTIVE                 0

6 rows selected.
```

从输出看出表空间 USERS 的两个数据文件 4 和 7 的 STATUS 为 NOT ACTIVE，说明已经结束了备份模式。

将当前重做日志文件信息写入归档日志文件。

例子 24-43　将当前重做日志文件信息写入归档日志文件

```
SQL> alter system archive log current;

System altered.
```

此时，将当前重做日志中的记录写到归档中，触发重做日志的切换并检查点信息。我们查看下当前的数据文件检查点信息。

例子 24-44　查看下当前的数据文件检查点信息

```
SQL>  select checkpoint_change#,file# from v$datafile;

CHECKPOINT_CHANGE#      FILE#
------------------ ----------
           1189326          1
           1189326          2
           1189326          3
           1189326          4
           1189326          7
           1189326          8

6 rows selected.
```

我们看到，此时的检查点都一致，并获得提升。



表空间热备时，如果该表空间涉及大量的 DML 操作，此时最好增大重做日志的大小，在联机备份时不要一次备份过多的表空间，选择在应用系统业务不繁忙的时段进行。

24.5　热备过程中对数据库崩溃的处理方法

在热备过程中如果发生数据库崩溃该如何处理，我们分析一下这个场景，以热备表空间 USERS 为例。此时数据文件头都锁定，备份正在复制文件而系统崩溃，显然表空间的数据文件文件头的 SCN 与控制文件中的不一致。此时数据库无法打开，会要求介质恢复，其实此时需要的是结束表空间的备份模式，解锁数据文件头 SCN。Oracle 会自动使用实例恢复，或者使用 recover datafile 指令实现数据文件的恢复（recover）。下面模拟这个过程。

启动表空间的备份模式，如下例所示。

例子 24-45　启动表空间的备份模式

```
SQL> alter tablespace users begin backup;

Tablespace altered.
```

查看表空间对应的数据文件的备份模式。

例子 24-46　查看表空间对应的数据文件的备份模式

```
SQL> select * from v$backup;

    FILE# STATUS                CHANGE# TIME
---------- ------------------ ---------- ---------
        1 NOT ACTIVE                  0
        2 NOT ACTIVE                  0
        3 NOT ACTIVE                  0
        4 ACTIVE                1189603 31-AUG-11
        7 ACTIVE                1189603 31-AUG-11
        8 NOT ACTIVE                  0

6 rows selected.
```

对应数据文件的 STATUS 参数值为 ACTIVE，说明数据文件处于备份模式，此时的数据文件头 SCN 为 1189603。

我们继续查看数据文件的起始检验点。

例子 24-47　查看数据文件的起始检验点

```
SQL> select checkpoint_change#,file# from v$datafile;

CHECKPOINT_CHANGE#      FILE#
------------------ ----------
         1189326          1
         1189326          2
         1189326          3
         1189603          4
         1189603          7
         1189326          8

6 rows selected.
```

数据文件 1、2、3、8 的检验点为 1189326，而数据文件 4、7 也就是我们设置为备份模式的表空间 USERS 对应的数据文件的检验点 SCN 为 1189603，比其他 4 个文件大。也就是比其他四个文件的文件头 SCN 新。原因是一旦使用 alter tablespace users begin backup 就会触发检验点事件，刷新数据文件的文件头 SCN，并冻结该 SCN。

接下来模拟一个实例失败的故障，然后重新数据库。

例子 24-48　重新数据库

```
SQL> shutdown abort;
ORACLE instance shut down.
SQL> startup
ORACLE instance started.

Total System Global Area  314572800 bytes
Fixed Size                 1219184 bytes
Variable Size             92276112 bytes
Database Buffers         218103808 bytes
Redo Buffers               2973696 bytes
```

```
Database mounted.
ORA-01113: file 4 needs media recovery
ORA-01110: data file 4: '/u01/app/oracle/oradata/PROD/users01.dbf'
```

此时报错了，首先发现数据文件 4 需要介质恢复。由于在表空间热备过程中实例故障，数据文件头依然锁定，数据文件头的 SCN 与控制文件中的 SCN 不同，此时数据库处于 MOUNT 状态，我们查询数据字典视图 v$backup 查询那些数据文件处于备份状态。

例子 24-49　查询那些数据文件处于备份状态

```
SQL> select * from v$backup;

     FILE# STATUS                CHANGE# TIME
---------- ------------------ ---------- ---------
         1 NOT ACTIVE                  0
         2 NOT ACTIVE                  0
         3 NOT ACTIVE                  0
         4 ACTIVE                1189603 31-AUG-11
         7 ACTIVE                1189603 31-AUG-11
         8 NOT ACTIVE                  0

6 rows selected.
```

显然，数据文件 4 和 7 处于备份模式，这两个文件都需要介质恢复，不过由于在控制文件中数据文件 4 的记录位置靠前，所以 Oracle 首先提示，数据文件 4 需要介质恢复。下面使用 recover 指令恢复该数据文件。

例子 24-50　使用 recover 指令恢复数据文件

```
SQL> recover datafile 4;
Media recovery complete.
```

在成功恢复数据文件 4 之后，我们尝试打开数据库，此时又出现了错误，要求数据文件 7 需要介质恢复，这在预料之中。

例子 24-51　尝试打开数据库

```
SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01113: file 7 needs media recovery
ORA-01110: data file 7: '/u01/app/oracle/oradata/PROD/users02.dbf'
```

接着我们恢复数据文件 7。

例子 24-52　恢复数据文件 7

```
SQL> recover datafile 7;
Media recovery complete.
```



上面两个数据文件的恢复也可以使用 alter tablespace users end backup 实现。

在成功恢复数据文件 4 和 7 之后，打开数据库，此时的数据文件已经是非备份模式。

例子 24-53　打开数据库，查看数据文件是否备份模式

```
SQL> alter database open;

Database altered.

SQL> select * from v$backup;

     FILE# STATUS             CHANGE# TIME
---------- ------------------ ---------- ---------
         1 NOT ACTIVE                 0
         2 NOT ACTIVE                 0
         3 NOT ACTIVE                 0
         4 NOT ACTIVE           1189603 31-AUG-11
         7 NOT ACTIVE           1189603 31-AUG-11
         8 NOT ACTIVE                 0

6 rows selected.
```

24.6　热备的原理

在介绍完如何热备后，对热备份有了直观的认识，接下来我们介绍热备的原理，从而更深刻地理解热备的每个步骤。

01 Oracle 的数据块大小是操作系统块大小的整数倍，直接复制数据文件到备份介质的过程中，由于数据库一直可用，数据块中不断地有数据进进出出，使用操作系统命令复制数据文件时，是基于操作系统块进行复制的，也就是说复制一个操作系统块的时候，首先锁定这个块，复制完成以后，解锁这个块，这样这个块在复制前和复制后是一致的。

但是对于一个 Oracle 数据库来说，一个块要分成几次进行复制，可能在复制一个操作系统数据块的时候，另外一个操作系统数据块被改变了，导致 Oracle 数据块在复制到备份介质以后，前一半是没有改动过的，后一半是改动过的，数据块的状态就不一致，数据块就不能够使用，备份也就不能够使用，这叫做分离数据块现象（split blocks）。

02 Oracle 在数据块的头部和尾部各存了一个版本号，通过这个来确认这个数据块是否是分离数据块，Oracle 修改过一个数据块以后，将这个数据块的版本号在头部和尾部各存放了一份。

03 为了修复分离数据块现象，我们在备份某个表空间的时候，首先进行 begin backup，通知数据库开始备份，在没有 end backup 命令之前，只要是进程修改了被备份的表空间所包含的数据块中的某条记录，Oracle 就会把该数据块中所包含的所有的数据行的数据生成重做记录，记录到日志文件中。

04 通过发出 begin backup 以后，第一次修改数据块中的数据行之前，在联机重做日志文件中

记录数据块的修改前的数据，这样在热备份恢复的时候，一旦发现某个数据块被分离，则会利用日志文件里的记录的数据对整个数据块进行恢复。

05 备份表空间中的所有的被修改过的数据块的所有数据行都被保存在了联机重做日志文件中，而不是只记录被修改的数据行的记录，因此在热备份过程中，会发现生成的联机重做日志文件的量比较大，这取决于业务的繁忙程度，例如 DML 量比较大，那么产生的日志会非常得多。

06 发出 begin backup 命令以后，Oracle 会对被备份的表空间所对应的数据文件触发文件级别的检查点进程，将内存中属于该表空间的所有脏数据块写入数据文件，检查点结束以后，数据文件头部（第一个和第一个数据块）的检查点 SCN 和日志序列号都不再变化，直至发出 end backup，数据文件头部的 SCN 和日志序列号才被更新。

07 在热备份的过程中，数据文件头部的 SCN 和日志序列号被冻结，但是数据文件本身还是最新的，和正常一样的更新和使用，DML 更新了某个表，这些更新都会被写入到数据文件中去。

08 冻结数据文件头部的 SCN 号和日志序列号，是为了将来使用热备份进行恢复的时候能够知道应该从哪里开始应用重做记录，也就是备份的数据文件头部所记录的 SCN 和日志序列号，定位到日志文件中，从日志文件中找到开始的 SCN，向后应用所有的日志文件。

09 End backup 以后，所有的数据文件的头部 SCN 和日志序列号推进到最新，因为数据文件本身就是最新的。

24.7　备份控制文件

控制文件除了冗余设计外，为了安全起见同样需要备份，控制文件中记录数据文件，重做日志文件的详细信息，一旦数据库的结构发生变化，如添加了表空间，增加了重做日志成员等控制文件都会变化，此时最好重新备份控制文件。

在现实的生产数据库中，控制文件损坏的可能性很小，如果真正实现了冗余设计，分布到不同的物理磁盘上，控制文件还是比较安全的。这里我们介绍两种备份控制文件的方法，DBA 可以通过手工操作备份控制文件。

1. 二进制备份

使用二进制备份将控制文件备份到指定的二进制文件，当控制文件丢失时，可以通过这个备份来恢复控制文件（如果数据库结构已经发生变化，则需要做深入的工作才能启动数据库）。

例子 24-54　通过备份恢复控制文件

```
SQL> alter database backup controlfile to '/u01/backup/controlfile.ctl.bak';

Database altered.

SQL> host ls /u01/backup
controlfile.ctl.bak  users01.dbf  users02.dbf
```

备份到 trace 文件，此时该 trace 文件位于 user_dump_dest 参数指定的目录下。

例子 24-55　备份到 trace 文件

```
SQL> alter database backup controlfile to trace;

Database altered.
```

我们确认用户的 DUMP 目录。

例子 24-56　确认用户的 DUMP 目录

```
SQL> show parameter user_dump_dest;

NAME                  TYPE      VALUE
---------------------------------------- --------------------------------
user_dump_dest             string        /u01/app/oracle/admin/PROD/udump
```

从输出知道我们备份的控制文件存放到 trace 文件中，该文件位于目录/u01/app/oracle/admin/PROD/下。下面我们查找该 trace 文件。

例子 24-57　查找 trace 文件

```
SQL> select spid from v$process where addr=
  2  (select paddr from v$session where sid=(select sid from v$mystat where
rownum=1) );

SPID
------------
7569

[oracle@ocm1 ~]$ cd /u01/app/oracle/admin/PROD/udump
[oracle@ocm1 udump]$ ll *7569*
-rw-r-----  1 oracle oinstall 9791 Aug 31 21:02 prod_ora_7569.trc
```

在找到该文件后，打开文件会发现在文件最后存在如下的一段记录。

例子 24-58　文件最后存在如下的一段记录

```
-- ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PERFORMANCE
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE "PROD" RESETLOGS  ARCHIVELOG
   MAXLOGFILES 10
   MAXLOGMEMBERS 5
   MAXDATAFILES 200
   MAXINSTANCES 1
   MAXLOGHISTORY 292
LOGFILE
  GROUP 1 (
    '/u01/app/oracle/oradata/PROD/disk1/redo01a.log',
    '/u01/app/oracle/oradata/PROD/disk2/redo01b.log'
  ) SIZE 100M,
  GROUP 2 (
    '/u01/app/oracle/oradata/PROD/disk2/redo02a.log',
    '/u01/app/oracle/oradata/PROD/disk3/redo02b.log'
  ) SIZE 100M,
```

```
  GROUP 3 (
    '/u01/app/oracle/oradata/PROD/disk3/redo03a.log',
    '/u01/app/oracle/oradata/PROD/disk4/redo03b.log'
  ) SIZE 100M,
  GROUP 4 (
    '/u01/app/oracle/oradata/PROD/disk4/redo04a.log',
    '/u01/app/oracle/oradata/PROD/disk5/redo04b.log'
  ) SIZE 100M,
  GROUP 5 (
    '/u01/app/oracle/oradata/PROD/disk5/redo05a.log',
    '/u01/app/oracle/oradata/PROD/disk1/redo05b.log'
  ) SIZE 100M
-- STANDBY LOGFILE
DATAFILE
  '/u01/app/oracle/oradata/PROD/system01.dbf',
  '/u01/app/oracle/oradata/PROD/undotbs01.dbf',
  '/u01/app/oracle/oradata/PROD/sysaux01.dbf',
  '/u01/app/oracle/oradata/PROD/users01.dbf',
  '/u01/app/oracle/oradata/PROD/users02.dbf',
  '/u01/app/oracle/oradata/PROD/audit01.dbf'
CHARACTER SET AL32UTF8
;
-- Configure RMAN configuration record 1
VARIABLE RECNO NUMBER;
EXECUTE :RECNO := SYS.DBMS_BACKUP_RESTORE.SETCONFIG('CONTROLFILE AUTOBACKUP','ON');
-- Commands to re-create incarnation table
-- Below log names MUST be changed to existing filenames on
-- disk. Any one log file from each branch can be used to
-- re-create incarnation records.
-- ALTER DATABASE REGISTER LOGFILE '/u01/arc3/1_1_743979786.dbf';
-- Recovery is required if any of the datafiles are restored backups,
-- or if the last shutdown was not normal or immediate.
RECOVER DATABASE USING BACKUP CONTROLFILE
-- Database can now be opened zeroing the online logs.
ALTER DATABASE OPEN RESETLOGS;
-- Commands to add tempfiles to temporary tablespaces.
-- Online tempfiles have complete space information.
-- Other tempfiles may require adjustment.
ALTER TABLESPACE TEMP ADD TEMPFILE '/u01/app/oracle/oradata/PROD/temp02.dbf'
     SIZE 104857600  REUSE AUTOEXTEND ON NEXT 8192  MAXSIZE 32767M;
ALTER TABLESPACE TEMP ADD TEMPFILE '/u01/app/oracle/oradata/PROD/temp01.dbf'
     SIZE 78643200  REUSE AUTOEXTEND ON NEXT 8192  MAXSIZE 32767M;
-- End of tempfile additions.
```

使用以上的控制文件创建脚本就可以重建控制文件。

2. 使用 RMAN 自动备份

在使用 RMAN 时，Oracle 允许控制文件自动备份，备份的触发条件是控制文件的内容发生变化，如创建了新的表空间。用户需要做的是修改 RMAN 参数，指定控制文件的存储目录。如下所示，先修改 RMAN 参数。

（1）设置 RMAN 的控制文件自动备份。

例子 24-59　设置 RMAN 的控制文件自动备份

```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;

old RMAN configuration parameters:
CONFIGURE CONTROLFILE AUTOBACKUP ON;
new RMAN configuration parameters:
CONFIGURE CONTROLFILE AUTOBACKUP ON;
new RMAN configuration parameters are successfully stored
```

（2）设置控制文件的备份目录。

例子 24-60　设置控制文件的备份目录

```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO
'/u01/backup/%F';

old RMAN configuration parameters:
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '/u01/backup/%F';
new RMAN configuration parameters:
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '/u01/backup/%F';
new RMAN configuration parameters are successfully stored
```

我们通过创建一个表空间，再确认是否自动备份了一个控制文件。

例子 24-61　创建一个表空间

```
SQL> create tablespace test
datafile '/u01/app/oracle/oradata/TEST/test01.dbf' size 100m;
```

此时，我们查看目录/u01/backup 下是否自动备份了控制文件。

例子 24-62　查看目录/u01/backup 下是否自动备份了控制文件

```
[oracle@ocm1 ~]$ ls /u01/backup
c-2058943077-20110910-00
```

通过查询，确定已经自动备份了控制文件，在以后需要备份的控制文件时需要使用 RMAN 来恢复。

24.8　介质恢复的原理

介质恢复是指数据文件损坏或者数据文件所在磁盘故障，而需要数据库恢复称为介质恢复，介质恢复包括两个步骤：

01 还原（restore）。该操作的目的是使用备份的文件替换损坏或丢失的文件，使用操作系统命令完成，将备份的文件复制到损坏的文件目录下，也可以复制到不同的磁盘下（此时需要修改控制文件中该文件的记录）。

02 恢复（recovery）。恢复是将归档的日志文件以及联机重做日志文件里的重做记录应用到还

原出来的文件上，该阶段使用 recover 命令完成

在打开数据库时，Oracle 需要读取控制文件，并比对控制文件以及数据文件中的 SCN 以确定是否需要恢复，此时数据文件、控制文件、联机重做日志文件必须同步，如果不同步，需要进行恢复，恢复成功以后才能打开数据库。下面详细介绍 SCN 的作用，SCN 是 Oracle 基于时间的一个函数值，作为系统的时间戳，这样就排除了使用本机时间造成的时间不一致问题。整个系统内的 SCN 是唯一的。

控制文件里记录了 3 种重要的检查点 SCN 号。

（1）系统检查点 SCN

当检查点进程启动时，会将检查点结束时间转化成 SCN 号记录在控制文件中，该检查点是全局范围内的，当发生文件级别的检查点时，不会更新该系统检查点 SCN。记录在控制文件中的系统检查点可以通过数据字典视图 v$database 实现，该视图就是从控制文件读取信息。

例子 24-63　查找系统检查点

```
SQL> select name,checkpoint change# from v$database;

NAME      CHECKPOINT_CHANGE#
------------------------------------------------------------------------
PROD                 1314470
```

从输出可以看出，当前数据库 PROD 的系统检查点为 1314470。

（2）数据文件检查点

当检查点进程启动时，包括全局范围以及文件级别的（将表空间设置为只读、begin backup、或将某个数据文件设置为 offline）检查点都会在控制文件里面记录每个数据文件上所发生的检查点 SCN。

将数据文件设置为正常离线、只读将表空间设置为 begin backup 时，触发文件级别的检查点并将该检查点更新控制文件和数据文件头部以后就不再变化，此时不会影响系统检查点。

文件级的检查点通过数据字典 v$datafile 查看，我们先查看下当前的文件检查点。

例子 24-64　查看当前的文件检查点

```
SQL> select file#,checkpoint_change#,last_change# from v$datafile;

    FILE# CHECKPOINT CHANGE# LAST CHANGE#
---------- ------------------ ------------
         1          1314470
         2          1314470
         3          1314470
         4          1314470
         7          1314470
         8          1314470

6 rows selected.
```

从输出看出数据文件的检查点 SCN 与系统检查点 SCN 相同，而 LAST_CHANGE#为空，它表示数据文件的结束 SCN。我们在下面会介绍。接下来触发一个文件级别检查点，将表空间 OFFLINE。

例子 24-65　将表空间 OFFLINE

```
SQL> alter tablespace users offline;

Tablespace altered.

SQL>  select file#,checkpoint change#,last change# from v$datafile;

     FILE# CHECKPOINT_CHANGE# LAST_CHANGE#
---------- ------------------ ------------
         1            1314470
         2            1314470
         3            1314470
         4            1314997      1314997
         7            1314997      1314997
         8            1314470

6 rows selected.
```

显然，此时的文件 SCN 为 1314997 比系统级 SCN 要高，说明表空间离线时，触发了文件级检查点，提升了该文件的 SCN。

（3）结束 SCN

每个数据文件都会有一个结束 SCN，在数据库正常运行期间，只要是在线的、可读写的数据文件，其终止 SCN 都为空，而对正常关闭数据库时，或者将数据文件正常离线、或只读时，都会由于触发了检查点进程，从而在控制文件里记录每个数据文件上的结束时的 SCN 号。

我们先查看数据库正常运行时的文件结束 SCN。先将表空间 USERS 设置为 ONLINE。

例子 24-66　将表空间 USERS 设置为 ONLINE

```
SQL> alter tablespace users online;

Tablespace altered.
```

此时，我们再查看文件检查点 SCN 信息。

例子 24-67　查看文件检查点 SCN 信息

```
SQL> select file#,checkpoint change#,last change# from v$datafile;

     FILE# CHECKPOINT CHANGE# LAST CHANGE#
---------- ------------------ ------------
         1            1314470
         2            1314470
         3            1314470
         4            1315469
         7            1315469
         8            1314470

6 rows selected.
```

从输出看出，当前的文件结束 SCN 为空，而文件 4 和 7 的初始检查点 SCN 为 1315469，比其

他文件的初始 SCN（1314470）要大，说明此时没有发生系统检查点。当数据库正常关闭时，所有文件的初始检查点会一致。下面我们将表空间 audit_tbs 设置为只读状态。

例子 24-68　将表空间 audit_tbs 设置为只读状态

```
SQL> alter tablespace audit tbs read only;

Tablespace altered.

SQL> select file#,checkpoint change#,last change# from v$datafile;

     FILE# CHECKPOINT CHANGE# LAST CHANGE#
---------- ------------------ ------------
         1            1314470
         2            1314470
         3            1314470
         4            1315945
         7            1315945
         8            1315991      1315991

6 rows selected.
```

我们再次查看文件的起始 SCN 和结束 SCN 时发现，数据文件 8 的起始 SCN 和结束 SCN 相同，因为我们将该文件对应的表空间设置为只读模式。

当我们结束该表空间的只读模式，改为读写模式后，该数据文件的结束 SCN 会重新为空，如下所示。

例子 24-69　表空间改为读写模式

```
SQL> alter tablespace audit tbs read write;

Tablespace altered.

SQL>  select file#,checkpoint_change#,last_change# from v$datafile;

     FILE# CHECKPOINT CHANGE# LAST CHANGE#
---------- ------------------ ------------
         1            1314470
         2            1314470
         3            1314470
         4            1315945
         7            1315945
         8            1316113

6 rows selected.
```

当将表空间 audit_tbs 设置为读写模式后，此时的起始 SCN 获得提升从 1315991 变为 1316113，而结束 SCN 为空。

正常关闭数据库，每个数据文件的结束 SCN 都是一致的（只读、离线的除外），在每个数据文件的头部，也记录了检查点 SCN，该检查点 SCN 与控制文件中记录的数据文件的检查点 SCN 是一致的。也就是说，全局范围和数据文件级别的检查点，不仅会将检查点 SCN 记录在控制文件

中，还会记录在数据文件的头部。下面查看数据文件头部的 SCN 信息。

例子 24-70　查看数据文件头部的 SCN 信息

```
SQL> select file#,checkpoint change# from v$datafile header;

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            1314470
         2            1314470
         3            1314470
         4            1315945
         7            1315945
         8            1316113

6 rows selected.
```

该信息与从控制文件中获取的SCN是一致的。从输出发现上述数据文件的检查点SCN不一致，是因为没有系统检查点被触发，下面我们出发一个系统检查点，将数据文件的起始 SCN 设置为一致。

例子 24-71　将数据文件的起始 SCN 设置为一致

```
SQL> alter system checkpoint;

System altered.

SQL>  select file#,checkpoint_change# from v$datafile;

     FILE# CHECKPOINT CHANGE#
---------- ------------------
         1            1316939
         2            1316939
         3            1316939
         4            1316939
         7            1316939
         8            1316939

6 rows selected.
```

从输出看出所有数据文件的检查点都一致了。接着我们查询系统全局检查点 SCN 的值，如下所示。

例子 24-72　查询系统全局检查点 SCN 的值

```
SQL> select checkpoint_change# from v$database;

CHECKPOINT CHANGE#
----------------------------------
             1316939
```

从输出看出和数据文件的起始检查点一致，数值一样。

数据库正常关闭时，会触发一个完全检查点，并用该检查点结束时的 SCN 号更新以上 4 个检

查点 SCN（除了离线和只读的）、系统检查点 SCN、数据文件检查点起始 SCN、数据文件检查点结束 SCN，以及数据文件头部的 SCN，这个 SCN 和数据文件起始 SCN 一致。当数据库打开时，如果发现这几个 SCN 的值不同就发生实例恢复，或要求介质恢复。

如果备份的数据文件覆盖当前的数据文件，则数据库会发现记录在控制文件里面的数据文件的检查点 SCN 号与从备份中还原的数据文件的头部记录的检查点 SCN 不一致，控制文件中的 SCN 比较新，备份中还原的数据文件头部的 SCN 比较旧，控制文件的 SCN 大于数据文件头部的 SCN，数据库不能打开，要求对还原的数据文件进行恢复，应用重做记录，从而将该数据文件头部的 SCN 提升到控制文件里记录的 SCN 号。

如果我们将备份的控制文件覆盖当前的控制文件，这时控制文件比较旧，而数据文件则是当前最新的数据文件，因此数据库会发现控制文件里面的 SCN 小于数据文件头部记录的 SCN，于是进行恢复。

启动数据库的时候，如果发现控制文件里记录的数据文件检查点 SCN 与数据文件头部记录的 SCN 相同，但是在每个在线的以及可读写的数据文件之间，他们的检查点 SCN 不同，那么会要求进行介质恢复。例如 begin backup 后，实例崩溃，重启数据库时就会发生这种情况，必须发出 end backup 命令才能打开数据库。

24.9　归档模式下的完全恢复

在掌握了热备份的方法、原理以及数据库的归档模式的区别，我们继续学习如何完成归档模式下的完全恢复。归档模式是 Oracle 推荐的一种数据库模式，以确保所有提交的数据不会丢失，以完成数据库的完全恢复。当然，在生产数据库的维护中有多方面的条件或者现实环境制约着完全恢复的可能性，如丢失部分归档文件等。但是对于 Oracle 而言，只要合理地使用归档模式，制定完善的归档模式下的备份和恢复方案，总能保证数据库在各种故障之下的完全恢复，完全恢复是我们研究的重点，也是学习的难点，本章将一一介绍。

在归档模式下面，如果控制文件和联机重做日志文件都没有损坏，而只是数据文件损坏，并且只要存在备份以及自从该备份以来所有的归档日志文件，就能够把数据库完全恢复到发生介质损坏的那个时间点上，通过 recover 命令实现恢复操作。

如果所有控制文件损坏，或者整个联机重做日志文件丢失，或者自从上次备份以来丢失了某个归档日志文件，则需要进行不完全恢复，因此对于控制文件、联机重做日志文件、归档日志文件，一定要注意保存好。

在模拟如何实现各种数据文件丢失情况下的完全恢复之前，我们先设定一下数据库的环境，首先将数据库设置为归档模式，然后设置归档目录，以便于使用自动搜索归档日志文件时，Oracle 可以自动发现需要的归档日志文件。

首先，设置归档目录。

例子 24-73　设置归档目录

```
SQL> alter system set log archive dest 1='location=/u01/arch1 mandatory';

System altered.
```

```
SQL> alter system set log_archive_dest_2='location=/u01/arch2';

System altered.
```

然后，查询确认归档模式。

例子 24-74　查询确认归档模式

```
SQL> archive log list;
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            /u01/arch2
Oldest online log sequence     1
Next log sequence to archive   3
Current log sequence           3
```

24.9.1　数据文件在有备份情况下的恢复

首先，我们对表空间 users 热备，为模拟恢复做准备。

例子 24-75　备份表空间 USERS 的数据文件

```
SQL> alter tablespace users begin backup;

Tablespace altered.
[oracle@ocm1 ~]$ cp /u01/app/oracle/oradata/TEST/users01.dbf  /u01/backup
```

然后创建一个表 t1，插入部分数据并提交。

例子 24-76　创建一个表 t1，插入数据并提交

```
SQL> create table t1 (id number) tablespace users;

Table created.
SQL> insert into t1 values(111);

1 row created.

SQL> commit;

Commit complete.
```

此时，在表 t1 的数据保存在 USERS 表空间中，而该表空间只有一个数据文件，我们删除该数据文件来模拟数据文件损坏。

```
[oracle@ocm1 ~]$ rm -rf /u01/app/oracle/oradata/TEST/users01.dbf
```

然后关闭数据库，重启数据库时发现数据文件故障。

例子 24-77　重启数据库时发现数据文件故障

```
SQL> startup
ORACLE instance started.
```

```
Total System Global Area  285212672 bytes
Fixed Size                  1218992 bytes
Variable Size              92276304 bytes
Database Buffers          188743680 bytes
Redo Buffers                2973696 bytes
Database mounted.
ORA-01157: cannot identify/lock data file 4 - see DBWR trace file
ORA-01110: data file 4: '/u01/app/oracle/oradata/TEST/users01.dbf'
```

此时，为了先将数据库打开（现实生产库中对外提供业务是最重要的，至少可以确保完好的表空间可以继续对外服务），先将该数据文件或数据文件对应的表空间离线。

例子 24-78　将该数据文件或数据文件对应的表空间离线

```
SQL> alter database datafile 4 offline;

Database altered.

SQL> alter database open;

Database altered.
```

此时，将备份的文件复制到损坏的数据文件所在目录，如果数据文件所在磁盘损坏可以复制到其他目录，但是需要使用 alter database rename datafile 修改控制文件中对该损坏数据文件的记录。

```
[oracle@ocm1 ~]$ cp /u01/backup/users01.dbf /u01/app/oracle/oradata/TEST
```

此时，我们已经完成了数据文件的 restore 任务，此时我们看看该数据文件为什么需要恢复，继续检查 SCN 信息。

首先通过数据字典 v$recover_file 查看哪些数据文件需要恢复。

例子 24-79　通过数据字典 v$recover_file 查看哪些数据文件需要恢复

```
SQL> l
  1* select * from v$recover_File
SQL> /

    FILE# ONLINE  ONLINE_  ERROR  CHANGE# TIME
----------------- ------------ ------------ -------------- ---------
        4 OFFLINE OFFLINE          505980 08-SEP-11
```

此时，提示数据文件 4 需要恢复，ERRO 类型为空，CHANGE#为 505980，说明了数据文件头中的 SCN 信息。

例子 24-80　查看数据文件信息

```
SQL> select file#,checkpoint change#,last change# from v$datafile where
file#=4;
    FILE#         CHECKPOINT_CHANGE# LAST_CHANGE#
---------------------------------------------------------------------------
        4         506888              506888
```

其中 CHECKPOINT_CHANGE#是控制文件中记录的数据文件 SCN。而 LAST_CHANGE#是数据文件离线时的数据文件 SCN。

显然数据文件头中记录的 SCN，和控制文件中记录的 SCN 不一致，需要恢复。下面执行尝试将表空间 USERS 设置为 ONLINE。

例子 24-81 尝试将表空间 USERS 设置为 ONLINE

```
SQL> alter tablespace users online;
alter tablespace users online
*
ERROR at line 1:
ORA-01113: file 4 needs media recovery
ORA-01110: data file 4: '/u01/app/oracle/oradata/TEST/users01.dbf'
```

提示数据文件 4 需要介质恢复，下面实现介质恢复。

例子 24-82 实现介质恢复

```
SQL> recover datafile 4;
Media recovery complete.
```

我们尝试查询表 t1 是否存在。

例子 24-83 尝试查询表 t1 是否存在

```
SQL> desc t1
 Name       Null?   Type
-------------------------------
 ID              NUMBER
```

从中看出表 t1 的定义存在，然后我们试图查询表 t1 中的数据。

例子 24-84 试图查询表 t1 中的数据

```
SQL> select * from t1;
select * from t1
            *
ERROR at line 1:
ORA-00376: file 4 cannot be read at this time
ORA-01110: data file 4: '/u01/app/oracle/oradata/TEST/users01.dbf'
```

提示说明数据文件 4 不能读，注意此时不是提示数据文件 4 不能锁定或者无法识别，说明我们已经 RESTORE 了数据文件，其实这里的提示说明数据文件需要 ONLINE。因为我们在打开数据库时，将数据文件设置为 OFFLINE 了。

例子 24-85 数据文件需要 ONLINE

```
SQL> alter database datafile 4 online;

Database altered.

SQL> select * from t1;

       ID
```

```
    ----------
          111
```

使用 DESC 指令查看表 t1 的定义时，虽然数据文件 4 无法读，但是依然可以查看表的定义，这里的演示过程是希望读者理解一个问题，就是我们恢复的是数据文件 4，而表定义是数据字典信息，这些信息是放在 SYSTEM 表空间中的，与我们需要恢复的数据文件无关。

至此，我们完全恢复了数据文件 4，此时也提升了控制文件和数据文件中的 SCN，使得二者一致。

例子 24-86　控制文件和数据文件中的 SCN 一致

```
    SQL>  select  file#,checkpoint_change#,last_change#  from  v$datafile  where
file#=4;

        FILE# CHECKPOINT_CHANGE# LAST_CHANGE#
    ------------------------------------------------ -------------------------
    507146
    SQL>  select file#,checkpoint change# from v$datafile header where file#=4;

        FILE# CHECKPOINT_CHANGE#
    ---------------- -------------------------------------
            4            507146
```

24.9.2　数据文件在无备份情况下的恢复

案例和上节相同，就是在数据文件丢失情况下的恢复，不同的是我们没有数据文件的备份，此时只有重建一个数据文件，如果保存了该表空间自从创建以来的所有归档日志，则可以完全恢复。

为了和上节的例子有所区别，我们新创建一个表空间并创建一个表，插入数据并提交。

例子 24-87　创建一个表空间

```
    SQL> create tablespace test
      2  datafile '/u01/app/oracle/oradata/TEST/test01.dbf' size 50m;

    Tablespace created.
```

创建一个表并插入数据，数据存储在新建的表空间 test 中。

例子 24-88　创建一个表并插入数据

```
    SQL> create table tsb (id number,name varchar2(20)) tablespace test;

    Table created.

    SQL> begin for i in 1..100 loop
      2    insert into tsb values(i,'name'||i);
      3   end loop;
      4  end;
      5  /

    PL/SQL procedure successfully completed.
```

```
SQL> commit;

Commit complete.

SQL> select count(*) from tsb;

  COUNT (*)
------------------
      100
```

我们没有对该数据文件进行任何形式的备份，下面删除该数据文件，模拟数据文件损坏。

```
[oracle@ocm1 ~]$ rm -rf /u01/app/oracle/oradata/TEST/test01.dbf
```

下面，我们清除 buffer cache 中的数据，然后查询表 tsb 中的数据，看如何报错。

例子 24-89　查询表 tsb 中的数据

```
SQL> alter system flush buffer_cache;

System altered.

SQL> select * from tsb;
select * from tsb
              *
ERROR at line 1:
ORA-01116: error in opening database file 6
ORA-01110: data file 6: '/u01/app/oracle/oradata/TEST/test01.dbf'
ORA-27041: unable to open file
Linux Error: 2: No such file or directory
Additional information: 3
```

提示数据文件 6 无法打开，Linux 没有这个文件。下面我们演示如何恢复没有备份的数据文件。首先将数据文件 6 OFFLINE。

例子 24-90　将数据文件 6 OFFLINE

```
SQL> alter database datafile 6 offline;

Database altered.
```

其次创建一个新的数据文件，目录和文件名都相同。

例子 24-91　创建一个新的数据文件

```
SQL> alter database create datafile
'/u01/app/oracle/oradata/TEST/test01.dbf';

Database altered.
```

下面，我们尝试将数据文件 ONLINE ，看提示什么。

例子 24-92　将数据文件 ONLINE

```
SQL> alter database datafile 6 online;
alter database datafile 6 online
```

```
*
ERROR at line 1:
ORA-01113: file 6 needs media recovery
ORA-01110: data file 6: '/u01/app/oracle/oradata/TEST/test01.dbf'
```

显然，如我们所料需要介质恢复，这里恢复的过程就是讲归档日志或者当前重做日志中的数据应用到数据文件中去。下面恢复数据文件并将数据文件 ONLINE。

例子 24-93　恢复数据文件并将数据文件 ONLINE

```
SQL> recover datafile 6;
Media recovery complete.
SQL> alter database datafile 6 online;

Database altered.
```

下面查看表 tsb 是否被恢复。

例子 24-94　查看表 tsb 是否被恢复

```
SQL> select count(*) from tsb;

  COUNT(*)
----------
       100
```

输出说明，我们成功恢复了没有备份的数据文件，切记成功恢复的关键是启动了数据库的归档模式，并且自该表空间创建后，所有插入的数据被重做日志或者归档日志保护，最终通过 RECOVER 指令得以完全恢复。

24.9.3　系统表空间数据文件损坏的完全恢复

系统表空间即 SYSTEM 表空间是十分重要的表空间，它存储了数据字典信息，如果没有系统表空间无论如何数据库无法启动。下面我们模拟系统表空间数据文件丢失如何恢复，其实恢复步骤与普通表空间数据文件丢失大致相同，包括 RESTORE 和 RECOVER 两个过程。首先我们备份系统表空间。

例子 24-95　备份系统表空间

```
SQL> alter tablespace system end backup;
[oracle@ocm1 ~]$ cp /u01/app/oracle/oradata/TEST/system01.dbf /u01/backup
SQL> alter tablespace system end backup;
```

然后启动数据库。

例子 24-96　启动数据库

```
SQL> startup
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                 1218992 bytes
```

```
Variable Size             71304784 bytes
Database Buffers         209715200 bytes
Redo Buffers               2973696 bytes
Database mounted.
ORA-01157: cannot identify/lock data file 1 - see DBWR trace file
ORA-01110: data file 1: '/u01/app/oracle/oradata/TEST/system01.dbf'
```

我们尝试将数据文件 OFFLINE，再打开数据库，看是否可以打开。

例子 24-97　尝试将数据文件 OFFLINE，再打开数据库

```
SQL> alter database datafile 1 offline;

Database altered.

SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01147: SYSTEM tablespace file 1 is offline
ORA-01110: data file 1: '/u01/app/oracle/oradata/TEST/system01.dbf'
```

显然，SYSTEM 表空间 OFFLINE 时，数据库无法打开。需要使用备份进行恢复。此时数据库处于 MOUNT 状态。下面复制备份文件。

```
[oracle@ocm1 ~]$ cp /u01/backup/system01.dbf /u01/app/oracle/oradata/TEST/
```

由于刚才我们将数据文件 1 离线，为了恢复的需要必须将其 ONLINE，如例子 24-98 所示。

例子 24-98　将数据文件 1 ONLINE

```
SQL> alter database datafile 1 online;

Database altered.
```

然后尝试打开数据库。

例子 24-99　尝试打开数据库

```
SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01113: file 1 needs media recovery
ORA-01110: data file 1: '/u01/app/oracle/oradata/TEST/system01.dbf'


SQL> recover datafile 1;
Media recovery complete.
```

提示数据文件 1 需要介质恢复，然后就可以打开数据库了。

例子 24-100　打开数据库

```
SQL> alter database open;
```

```
Database altered.
```

24.9.4　当前 UNDO 表空间损坏的完全恢复

UNDO 表空间损坏，会造成回滚数据丢失，比如用户启动了闪回，此时就无法使用 UNDO 中的数据实现闪回查询，无法恢复用户误删除的数据。下面我们演示如何恢复 UNDO 表空间。首先查询当前 UNDO 的信息。

例子 24-101　查询当前 UNDO 的信息

```
SQL> show parameter undo

NAME                                 TYPE        VALUE
------------------------------------ -----------------------------
undo_management                      string      AUTO
undo_retention                       integer     900
undo_tablespace                      string      UNDOTBS1
```

查询相关数据文件信息。

例子 24-102　查询相关数据文件信息

```
SQL>    select    file_name,tablespace_name,online_status,status    from
dba_data_files
  2  where tablespace_name like 'UNDO%';

FILE_NAME                              TABLESPACE_NAME ONLINE  STATUS
--------------------------------------------------------------------------
/u01/app/oracle/oradata/TEST/undotbs01.dbf  UNDOTBS1      ONLINE  AVAILABLE
/u01/app/oracle/oradata/TEST/temp02.dbf     UNDOTBS1      ONLINE  AVAILABLE
```

我们先热备这两个数据文件。

例子 24-103　先热备这两个数据文件

```
SQL> alter tablespace undotbs1 begin backup;

Tablespace altered.

SQL> host
[oracle@ocm1 ~]$ cp /u01/app/oracle/oradata/TEST/undotbs01.dbf /u01/backup
[oracle@ocm1 ~]$ cp /u01/app/oracle/oradata/TEST/temp02.dbf /u01/backup
[oracle@ocm1 ~]$ exit
exit

SQL> alter tablespace undotbs1 end backup;

Tablespace altered.
```

下面我们删除这两个数据文件，模拟当前的还原表空间损坏。

例子 24-104　模拟当前的还原表空间损坏

```
[oracle@ocm1 TEST]$ rm -rf undotbs01.dbf
```

```
[oracle@ocm1 TEST]$ rm -rf temp02.dbf
```

下面创建一个新的用户，并使用该新用户连接数据库，并创建一个表，测试还原表空间是否可用。

创建一个用户。

例子 24-105 创建一个用户

```
SQL> create user udo identified by oracle;

User created.

SQL> grant resource,connect to udo;

Grant succeeded.
```

连接数据库并创建一个表。

例子 24-106 连接数据库并创建一个表

```
SQL> create table ut (id number);
create table ut (id number)
*
ERROR at line 1:
ORA-01116: error in opening database file 2
ORA-01110: data file 2: '/u01/app/oracle/oradata/TEST/undotbs01.dbf'
ORA-27041: unable to open file
Linux Error: 2: No such file or directory
Additional information: 3
```

此时报错，因为需要还原段但是我们已经删除了该数据文件，非 SYS 用户无法使用系统还原段，所以必须恢复该数据文件。

启动数据库到 MOUNT 状态，然后使用热备的备份恢复。

例子 24-107 启动数据库到 MOUNT 状态

```
SQL> startup mount;
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                  1218992 bytes
Variable Size              71304784 bytes
Database Buffers          209715200 bytes
Redo Buffers                2973696 bytes
Database mounted.
```

将备份文件复制到原来目录。

例子 24-108 将备份文件复制到原来目录

```
[oracle@ocm1 ~]$ cp /u01/backup/undotbs01.dbf /u01/app/oracle/oradata/TEST/
[oracle@ocm1 ~]$ cp /u01/backup/temp02.dbf  /u01/app/oracle/oradata/TEST/
```

接下来恢复两个数据文件。

例子 24-109　恢复两个数据文件

```
SQL> recover datafile 2;
Media recovery complete.
SQL> recover datafile 7;
Media recovery complete.
```

然后打开数据库。

例子 24-110　打开数据库

```
SQL> alter database open;

Database altered.
```

最后查看还原表空间以及数据文件的状态信息。

例子 24-111　查看还原表空间以及数据文件的状态信息

```
SQL>     select     file name,tablespace name,online status,status     from
dba Data files
  2  where tablespace_name like 'UNDO%';

FILE NAME                          TABLESPACE NAME  ONLINE   STATUS
-------------------------------------------------------- ------- ---------
/u01/app/oracle/oradata/TEST/undotbs01.dbf   UNDOTBS1     ONLINE  AVAILABLE
/u01/app/oracle/oradata/TEST/temp02.dbf     UNDOTBS1      ONLINE  AVAILABLE
```

我们知道还原表空间的几个作用即读一致性、实例恢复和事务回滚，所以如果还原表空间损坏会造成无法实现这些功能，使得数据库无法使用。如果将数据库处于归档模式，并做了合理的备份，那么任何数据文件的损坏或者丢失都不是问题，使用备份文件、当前重做日志和归档重做日志就可以恢复到数据库的最新状态而不会丢失任何数据。

24.9.5　非当前 UNDO 表空间损坏的完全恢复

我们知道还原表空间可以创建多个，但是当前的还原表空间只有一个，本节我们讨论如何恢复非当前还原表空间。通过如下的具体步骤演示如何恢复非当前还原表空间。

先创建一个还原表空间。

例子 24-112　创建一个还原表空间

```
SQL> create undo tablespace undotbs2
  2  datafile '/u01/app/oracle/oradata/TEST/undotbs2.dbf' size 100m
  3  extent management local;

Tablespace created.
```

列出与当前的还原表空间相关的参数信息。

例子 24-113　列出与当前的还原表空间相关的参数信息

```
SQL> show parameter undo
```

```
NAME                                 TYPE        VALUE
------------------------------------------------------------ -----------
undo management                      string      AUTO
undo retention                       integer     900
undo_tablespace                      string      UNDOTBS1
```

我们已经创建了一个还原表空间 UNDOTBS2，但是当前的还原表空间依然为 UNDOTBS1，如果需要可以将表空间 UNDOTBS2 设置为当前的还原表空间，比如还原表空间 UNDOTBS1 损坏的情况下。

下面，我们继续列出和还原表空间对应的数据文件信息以及状态信息。

例子 24-114　列出和还原表空间对应的数据文件信息以及状态信息

```
SQL>    select    file name,tablespace name,online status,status    from
dba data files
    where tablespace_name like 'UNDO%';

FILE NAME                            TABLESPACE NAME ONLINE  STATUS
----------------------------------------------------------- ------- ----------
/u01/app/oracle/oradata/TEST/undotbs01.dbf  UNDOTBS1     ONLINE  AVAILABLE
/u01/app/oracle/oradata/TEST/temp02.dbf     UNDOTBS1     ONLINE  AVAILABLE
/u01/app/oracle/oradata/TEST/undotbs2.dbf   UNDOTBS2     ONLINE  AVAILABLE

8 rows selected.
```

下面我们模拟非当前还原表空间数据文件损坏后如何恢复。首先，关闭数据库，然后使用操作系统指令删除掉该表空间对应的数据文件。

例子 24-115　删除掉表空间对应的数据文件

```
[oracle@ocm1 ~]$ cd /u01/app/oracle/oradata/TEST
[oracle@ocm1 TEST]$ ls
control01.ctl  example01.dbf  redo03.log   temp01.dbf  undotbs01.dbf
control02.ctl  redo01.log     sysaux01.dbf temp02.dbf  undotbs2.dbf
control03.ctl  redo02.log     system01.dbf test01.dbf  users01.dbf
[oracle@ocm1 TEST]$ rm -f undotbs2.dbf
```

下面重启数据库。

例子 24-116　重启数据库

```
SQL> startup
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                  1218992 bytes
Variable Size              71304784 bytes
Database Buffers          209715200 bytes
Redo Buffers                2973696 bytes
Database mounted.
ORA-01157: cannot identify/lock data file 8 - see DBWR trace file
ORA-01110: data file 8: '/u01/app/oracle/oradata/TEST/undotbs2.dbf'
```

提示无法识别或锁定数据文件 8，也就是我们刚刚删除的还原数据文件，此时就需要恢复，我们的思路是重建该表空间，所以先设法打开数据库，此时我们需要先将数据文件 8 离线，然后打开数据库。

例子 24-117　将数据文件 8 离线，然后打开数据库

```
SQL> alter database datafile 8 offline;

Database altered.

SQL> alter database open;

Database altered.
```

我们继续验证当前还原表空间的信息。

例子 24-118　验证当前还原表空间的信息

```
SQL>    select    file_name,tablespace_name,online_status,status    from
dba data files
  2  where tablespace name like 'UNDO%';

FILE_NAME                                    TABLESPACE_NAME ONLINE_ STATUS
-------------------------------------------------------- ------- ----------
/u01/app/oracle/oradata/TEST/undotbs01.dbf   UNDOTBS1      ONLINE  AVAILABLE
/u01/app/oracle/oradata/TEST/temp02.dbf      UNDOTBS1      ONLINE  AVAILABLE
/u01/app/oracle/oradata/TEST/undotbs2.dbf    UNDOTBS2      OFFLINE AVAILABLE
```

我们发现表空间 UNDOTBS2 的数据文件依然记录在数据字典中，这样当我们重建该表空间时由于重名就会报错，此时，我们使用 drop tablespace 指令删除掉该表空间。继续查看删除结果。

例子 24-119　查看删除结果

```
SQL>    select    file name,tablespace name,online status,status    from
dba data files
  2  where tablespace name like 'UNDO%';

FILE_NAME                                    TABLESPACE_NAME ONLINE_ STATUS
---------------------------------------------------- ------- ----------
/u01/app/oracle/oradata/TEST/undotbs01.dbf   UNDOTBS1      ONLINE  AVAILABLE
/u01/app/oracle/oradata/TEST/temp02.dbf      UNDOTBS1      ONLINE  AVAILABLE
```

目前为止，我们彻底清楚了还原表空间 UNDOTBS2 的信息，接下来就通过重建的方式恢复这个非当前的还原表空间。

例子 24-120　通过重建的方式恢复这个非当前的还原表空间

```
SQL> create undo tablespace undotbs2
  2  datafile '/u01/app/oracle/oradata/TEST/undotbs2.dbf' size 100m
  3  extent management local;

Tablespace created.
```

我们查看是否成功创建该非当前的还原表空间。

例子 24-121　查看是否成功创建该非当前的还原表空间

```
    SQL>select       file_name,tablespace_name,online_status,status       from
dba_data_files
      2*  where tablespace_name like 'UNDO%'

    FILE_NAME                               TABLESPACE_NAME ONLINE  STATUS
    ---------------------------------------------    ---------------    -------
----------
    /u01/app/oracle/oradata/TEST/undotbs01.dbf  UNDOTBS1     ONLINE  AVAILABLE
    /u01/app/oracle/oradata/TEST/temp02.dbf     UNDOTBS1     ONLINE  AVAILABLE
    /u01/app/oracle/oradata/TEST/undotbs2.dbf    UNDOTBS2     ONLINE  AVAILABLE
```

在生产库中，非当前还原表空间是必要的，因为一旦当前还原表空间损坏，则切换起来就很快速，最大程度减少了对系统业务的影响。

24.10　何时使用不完全恢复

不完全恢复是一种迫不得已的恢复方式，所谓的不完全就是数据库不能恢复到最新的状态，会有数据丢失。下面总结一下不完全恢复的场合。

24.10.1　不完全恢复的场合

不完全恢复的场合如下：

- 在应用归档日志过程中，丢失了一个或多个归档日志文件，则只能将数据库恢复到最后丢失的那个日志文件为止。
- 丢失了当前正在使用的日志文件或者丢失了没有归档的非当前日志文件。
- 使用备份的控制文件完成恢复。
- 用户误操作，如误删除了某个重要的表，可以使用不完全恢复将数据库恢复到误操作之前的时间点。

不完全恢复也是建立在备份的基础上，并且也需要使用部分归档日志文件。在不完全恢复之前要确认有备份文件，并且该文件的备份在要恢复的时间点之前完成，具有从备份到要恢复的时间点之间所有的归档日志。

24.10.2　不完全恢复的类型

下面介绍 3 种类型的不完全恢复。

（1）基于时间点的不完全恢复（time-based），恢复到指定的、历史上的某个时间点，这个时间点在恢复时不好把握，需要在恢复过程中调整时间，并检查是否恢复到满意的程度而停止不完全恢复。指令如下所示。

```
    recover database until time 'yyyy-mm-dd hh24:mi:ss';
```

（2）基于撤销的不完全恢复（cancel-based），恢复到我们输入 cancel 命令时为止，这通常用于丢失联机重做日志文件或归档日志文件，当恢复到丢失的日志文件的时候，只能输入 cancel 命令，表示恢复到此为止。指令如下所示。

```
recover database until cancel
```

（3）基于 SCN 号的不完全恢复（change-based），恢复到我们指定的、历史上的某个 SCN，从本质上来说，这与基于时间的不完全恢复一样，除非我们能够确定要恢复到 SCN 号，建议使用基于时间点的恢复。指令如下所示。

```
recover database until cancel scn <integer>
```

建议在进行不完全恢复之前，将整个数据库进行冷备份，备份所有的数据文件、控制文件、联机重做日志文件，万一恢复失败，能够将现场彻底还原到进行不完全恢复之前的状态或者至少归档当前的联机日志文件、备份当前控制文件。

24.11　所有控制文件丢失的恢复方法

Oracle 强烈建议将控制文件冗余存放，这是保证数据库安全的重要举措，如果冗余中的一个控制文件损坏，可以通过复制没有损坏的一个控制文件即可恢复，多个控制文件中的一个丢失不影响数据库的运行，但是会记录在告警日志中。

如果全部控制文件丢失，则需要恢复，否则数据库无法启动。此时需要使用备份的控制文件或者使用脚本创建新的控制文件。下面我们研究如何恢复所有控制文件丢失的情况。

24.11.1　使用备份的控制文件

如果数据库的所有控制文件丢失，但是有控制文件的备份，并且数据文件、重做日志文件以及归档日志文件都存在，就不会丢失数据。下面我们模拟所有控制文件丢失，看如何使用备份的控制文件实现完全恢复。

首先创建两个表，目的是测试使用备份的控制文件恢复后，验证这些表都存在，即没有数据丢失。我们在备份控制文件之前创建一个表，在备份文件之后也创建一个表。

创建表 t1。

例子 24-122　创建表 t1

```
SQL> create table t1 tablespace users as select * from scott.emp;

Table created.
```

备份控制文件。

例子 24-123　备份控制文件

```
SQL> alter database backup controlfile to '/u01/backup/control.ctl.bak';

Database altered.
```

创建表 t2。

例子 24-124　创建表 t2

```
SQL> create table t2 tablespace users as select * from scott.dept;

Table created.
```

接着我们模拟这个故障，关闭数据库将所有控制文件删除。生产库中如果所有控制文件损坏数据库会立即关闭。

例子 24-125　将所有控制文件删除

```
[oracle@ocm1 TEST]$ ls
control01.ctl  example01.dbf  redo03.log    temp01.dbf  undotbs01.dbf
control02.ctl  redo01.log     sysaux01.dbf  temp02.dbf  users01.dbf
control03.ctl  redo02.log     system01.dbf  test01.dbf
[oracle@ocm1 TEST]$ rm -rf *.ctl;
```

接着启动数据库。

例子 24-126　启动数据库

```
SQL> startup
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                  1218992 bytes
Variable Size              71304784 bytes
Database Buffers          209715200 bytes
Redo Buffers                2973696 bytes
ORA-00205: error in identifying control file, check alert log for more info
```

提示无法识别控制文件，即在参数文件中指定的位置不能发现控制文件，因为我们删除了所有控制文件，所以这个提示在预料之中。

此时数据库处于 NOMOUNT 状态，我们将备份的控制文件复制到指定目录。

例子 24-127　将备份的控制文件复制到指定目录

```
[oracle@ocm1 ~]$cp /u01/backup/control.ctl.bak /u01/app/oracle/oradata/TEST/control01.ctl
[oracle@ocm1 ~]$cp /u01/backup/control.ctl.bak /u01/app/oracle/oradata/TEST/control02.ctl
[oracle@ocm1 ~]$cp /u01/backup/control.ctl.bak /u01/app/oracle/oradata/TEST/control03.ctl
```

接着我们启动数据库到 MOUNT 状态，查看控制文件和数据文件中的 SCN，看是否需要恢复，以提升控制文件的 SCN。

例子 24-128　查看控制文件和数据文件中的 SCN

```
SQL> select file#,checkpoint change# from v$datafile;

    FILE# CHECKPOINT_CHANGE#
```

```
---------- ------------------
         1            711484
         2            711484
         3            711484
         4            711484
         5            711484
         6            711484
         7            711484

7 rows selected.
```

例子 24-129　查看控制文件和数据文件中的 SCN

```
SQL> select file#,checkpoint_change# from v$datafile_header;

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            713762
         2            713762
         3            713762
         4            713762
         5            713762
         6            713762
         7            713762

7 rows selected.
```

我们发现当前控制文件的 SCN 为 711484，而当前数据文件头中的 SCN 为 713762，显然控制文件中的 SCN 要旧，需要通过归档或者当前日志文件的记录来提升 SCN。

下面我们使用备份的控制文件实现数据库恢复。

例子 24-130　使用备份的控制文件实现数据库恢复

```
SQL> recover database using backup controlfile;
ORA-00279: change 712079 generated at 09/12/2011 09:27:20 needed for thread
1
ORA-00289: suggestion : /u01/arch2/1_10_761570828.dbf
ORA-00280: change 712079 for thread 1 is in sequence #10


Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
auto
ORA-00308: cannot open archived log '/u01/arch2/1_10_761570828.dbf'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
Additional information: 3


ORA-00308: cannot open archived log '/u01/arch2/1_10_761570828.dbf'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
Additional information: 3
```

此时使用完了归档日志，接着再使用当前的重做日志，至于哪个日志文件是当前的重做日志，可以通过尝试实现，如例子 24-131 所示。

例子 24-131　尝试实现当前的重做日志

```
SQL> recover database using backup controlfile;
ORA-00279: change 712079 generated at 09/12/2011 09:27:20 needed for thread
1
ORA-00289: suggestion : /u01/arch2/1_10_761570828.dbf
ORA-00280: change 712079 for thread 1 is in sequence #10

Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
/u01/app/oracle/oradata/TEST/redo01.log
ORA-00328: archived log ends at change 691254, need later change 712079
ORA-00334: archived log: '/u01/app/oracle/oradata/TEST/redo01.log'
```

上面指定的日志文件显然不是当前的重做日志，接下来我们使用重做日志组 2 尝试。

例子 24-132　使用重做日志组 2 尝试

```
SQL>  recover database using backup controlfile;
ORA-00279: change 712079 generated at 09/12/2011 09:27:20 needed for thread
1
ORA-00289: suggestion : /u01/arch2/1_10_761570828.dbf
ORA-00280: change 712079 for thread 1 is in sequence #10

Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
/u01/app/oracle/oradata/TEST/redo02.log
Log applied.
Media recovery complete.
```

这次尝试成功，提示介质恢复完毕，说明重做日志组 2 是当前的重做日志组。接着使用 RESETLOGS 打开数据库，使数据库的日志文件从新的序列号开始计数。

例子 24-133　使用 RESETLOGS 打开数据库

```
SQL> alter database open resetlogs;

Database altered.
```

为了验证数据库是否丢失数据，我们查询表 t1 和 t2，看是否存在。

例子 24-134　查询表 t1 和 t2 确认是否存在

```
SQL> SQL> select count(*) from t1;

  COUNT(*)
----------
        14

SQL> select count(*) from t2;

  COUNT(*)
----------
```

```
             4
```

通过查询结果显示，表 t1 和 t2 的数据都没有丢失，至此，我们在没有丢失数据的情况下恢复了控制文件。

24.11.2　重建控制文件

恢复控制文件的第二种方式就是使用重建的方式，重建控制文件是不得已之选，在既没有备份也没有冗余的情况下使用重建的方式，但是使用重建需要一些数据库信息，这些信息可以存放在 TRACE 文件中。我们使用如下指令获得这些信息，并保存在指定的 TRACE 文件中。

例子 24-135　获得使用重建需要的一些数据库信息

```
SQL> alter database backup controlfile to trace;

Database altered.
```

我们到 user_dump_file 目录下寻找该文件（可通过时间排序找到），查看如下所示的内容。

例子 24-136　查看文件内容

```
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE "TEST" RESETLOGS  ARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 3
    MAXDATAFILES 100
    MAXINSTANCES 8
    MAXLOGHISTORY 292
LOGFILE
  GROUP 1 '/u01/app/oracle/oradata/TEST/redo01.log'  SIZE 50M,
  GROUP 2 '/u01/app/oracle/oradata/TEST/redo02.log'  SIZE 50M,
  GROUP 3 '/u01/app/oracle/oradata/TEST/redo03.log'  SIZE 50M
-- STANDBY LOGFILE
DATAFILE
  '/u01/app/oracle/oradata/TEST/system01.dbf',
  '/u01/app/oracle/oradata/TEST/undotbs01.dbf',
  '/u01/app/oracle/oradata/TEST/sysaux01.dbf',
  '/u01/app/oracle/oradata/TEST/users01.dbf',
  '/u01/app/oracle/oradata/TEST/example01.dbf',
  '/u01/app/oracle/oradata/TEST/test01.dbf',
  '/u01/app/oracle/oradata/TEST/temp02.dbf'
CHARACTER SET WE8ISO8859P1
;
```

上面这些数据就是创建控制文件的需要的信息和创建方式。下面我们模拟一些操作，创建一些表，并删除所有控制文件，使用重建的方式恢复数据库。

例子 24-137　创建表 t21

```
SQL> create table t21 tablespace users as select * from scott.emp;

Table created.
```

关闭数据库后删除所有控制文件，然后重启数据库提示如下。

例子 24-138　重启数据库

```
SQL> startup
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                  1218992 bytes
Variable Size              71304784 bytes
Database Buffers          209715200 bytes
Redo Buffers                2973696 bytes
ORA-00205: error in identifying control file, check alert log for more in
```

此时，数据库为 NOMOUNT 模式，我们使用以上提供的脚本来重建控制文件。

例子 24-139　重建控制文件

```
SQL> CREATE CONTROLFILE REUSE DATABASE "TEST" RESETLOGS  ARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 3
    MAXDATAFILES 100
    MAXINSTANCES 8
    MAXLOGHISTORY 292
LOGFILE
  GROUP 1 '/u01/app/oracle/oradata/TEST/redo01.log'  SIZE 50M,
  GROUP 2 '/u01/app/oracle/oradata/TEST/redo02.log'  SIZE 50M,
  GROUP 3 '/u01/app/oracle/oradata/TEST/redo03.log'  SIZE 50M
-- STANDBY LOGFILE
DATAFILE
  '/u01/app/oracle/oradata/TEST/system01.dbf',
  '/u01/app/oracle/oradata/TEST/undotbs01.dbf',
  '/u01/app/oracle/oradata/TEST/sysaux01.dbf',
  '/u01/app/oracle/oradata/TEST/users01.dbf',
  '/u01/app/oracle/oradata/TEST/example01.dbf',
  '/u01/app/oracle/oradata/TEST/test01.dbf',
  '/u01/app/oracle/oradata/TEST/temp02.dbf'
CHARACTER SET WE8ISO8859P1
;
Control file created.
```

此时提示成功创建控制文件。由于是重建了控制文件，控制文件的系统 SCN 为 0，这个数据是从当前日志文件的 first_change#获得的，如例子 24-140 所示。

例子 24-140　从当前日志文件的 first_change#获得控制文件的 SCN

```
SQL> select checkpoint change# from v$database;

CHECKPOINT_CHANGE#
------------------
                 0
```

这个 CHECKPOINT_CHANGE#就是从如下的 first_change#获得的。

例子 24-141　获得 CHECKPOINT_CHANGE#

```
SQL> select group#,first_change#,status from v$log;

   GROUP# FIRST_CHANGE# STATUS
---------- ------------- ----------------
        1             0 UNUSED
        3             0 CURRENT
        2             0 UNUSED
```

而此时的数据文件 SCN 是从数据文件头部获取的，如下查询所示。

例子 24-142　从数据文件头部获取 SCN

```
SQL> select file#,checkpoint_change# from v$datafile;

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
        1            717465
        2            717465
        3            717465
        4            717465
        5            717465
        6            717465
        7            717465

7 rows selected.
```

控制文件中的数据文件 SCN（即 CHECKPOINT_CHANGE#）是从数据文件头部的 SCN 获得的。

例子 24-143　从数据文件头部获得数据文件 SCN

```
SQL> select file#,checkpoint_change# from v$datafile_header;

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
        1            717465
        2            717465
        3            717465
        4            717465
        5            717465
        6            717465
        7            717465

7 rows selected.
```

从查询结果可知控制文件中的文件 SCN 与数据文件头部的 SCN 相同。

接下来恢复数据库。

例子 24-144　恢复数据库

```
SQL> recover database using backup controlfile;
ORA-00279: change 717465 generated at 09/12/2011 12:07:38 needed for thread
```

```
1
    ORA-00289: suggestion : /u01/arch2/1_1_761658765.dbf
    ORA-00280: change 717465 for thread 1 is in sequence #1


    Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
    auto
    ORA-00308: cannot open archived log '/u01/arch2/1 1 761658765.dbf'
    ORA-27037: unable to obtain file status
    Linux Error: 2: No such file or directory
    Additional information: 3


    ORA-00308: cannot open archived log '/u01/arch2/1 1 761658765.dbf'
    ORA-27037: unable to obtain file status
    Linux Error: 2: No such file or directory
    Additional information: 3
```

此时使用完了归档日志。接下来应用当前的重做日志文件。

例子 24-145　应用当前的重做日志文件

```
    SQL> recover database using backup controlfile;
    ORA-00279: change 717465 generated at 09/12/2011 12:07:38 needed for thread
1
    ORA-00289: suggestion : /u01/arch2/1 1 761658765.dbf
    ORA-00280: change 717465 for thread 1 is in sequence #1


    Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
    /u01/app/oracle/oradata/TEST/redo03.log
    Log applied.
    Media recovery complete.
```

打开数据库。

例子 24-146　打开数据库

```
    SQL> alter database open resetlogs;

    Database altered.
```

查询表 t21 是否存在。

例子 24-147　查询表 t21 是否存在

```
    SQL> desc t21;
     Name                                   Null?    Type
     ----------------------------------- ----------------------------
     EMPNO                                            NUMBER(4)
     ENAME                                            VARCHAR2(10)
     JOB                                              VARCHAR2(9)
     MGR                                              NUMBER(4)
     HIREDATE                                         DATE
     SAL                                              NUMBER(7,2)
```

```
COMM                                             NUMBER(7,2)
DEPTNO                                           NUMBER(2)
```

至此，成功重建控制文件，并且没有数据丢失。由于控制文件中没有包括临时表空间，所以手工添加临时表空间，此时我们手工添加该数据文件。

```
SQL>ALTER TABLESPACE TEMP
ADD TEMPFILE '/u01/app/oracle/oradata/TEST/temp01.dbf'
SIZE 20971520  REUSE AUTOEXTEND ON NEXT 655360  MAXSIZE 32767M;
```

至此，完成通过重建控制文件来恢复数据库的全部过程。

24.12 本章小结

本章介绍了手工管理的备份与恢复，首先介绍了备份恢复涉及的重要概念（如物理备份、逻辑备份、冷备、热备以及恢复）的含义。在非归档模式下介绍了冷备的步骤和恢复方法，然后介绍了热备的原理、热备步骤以及热备过程中对于数据库崩溃的处理方法。最后介绍了归档模式下的完全恢复的几种案例以及不完全恢复的场合。本章的内容希望读者多加实践，只要模仿书中的例子即可掌握手工备份恢复的方法。

第 25 章

OEM管理与使用

OEM 是 Oracle 提供的图形化管理工具，是 DBA 管理数据库的利器。Oracle 提供了两种产品，一个是 Oracle Enterprise Manager Database Control，即 OEM Database Control，它是针对单机版的管理工具；另一个是 Oracle Enterprise Manager Grid Control，即 OEM Grid Control，它可以管理多个数据库，而且同时可以监控和管理主机、应用服务器、WebLogic 等资源。可以说 OEM Database Control 是 OEM Grid Control 简化版本，功能较后者减少很多，但是对于 DBA 管理日常工作已经足够了。本章我们重点介绍 OEM 的管理和使用。

25.1　OEM架构

OEM 是单机版的 Oracle 数据库管理工具，图 25-1 是 OEM 的架构图，我们通过架构图分析一下 OEM 访问的过程，并分析相应组件。

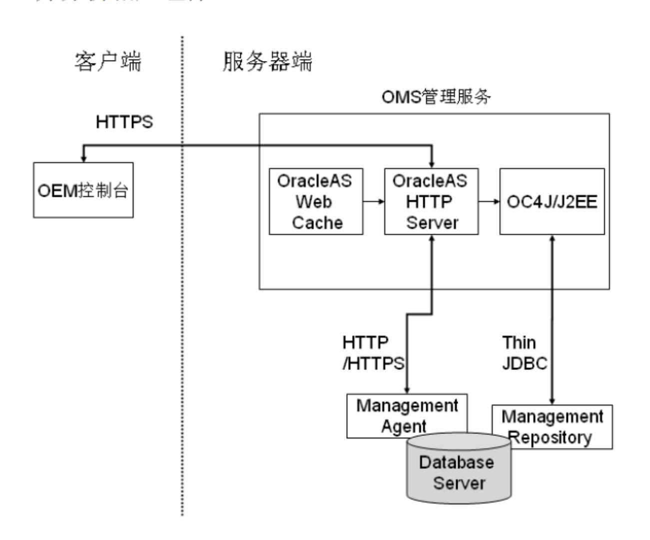


图 25-1 OEM 架构图

对于单机版的 OEM 不需要安装管理代理获取数据库服务器的信息，只有安装 OEM Grid Control 时才需要在要管理的数据库服务器端安装额外的 Agent 软件。下面是 OEM 架构中组件的作用。

- Oracle Management Repository：存储要监控的数据库的管理数据以及状态信息。一般需要再创建额外的数据库来存储，也可以在要管理的数据库上创建，但是需要考虑安全性，因为在 Reposigory 中是一组模式，所以在单独的数据库或者在药管理的数据库上存储都是允许的。
- OMS 管理服务：OMS 主要用于监控和管理安装了 Management Agent 的数据库，并负责将获得管理信息存入 Management Repository。它又包含 3 个组成部分。
- Oracle HTTP Server：Web 服务器，是内置的 Apache 服务器。
- OC4J/J2EE：它由 Java 编写，兼容了 J2EE。
- Oracle Web Cache：用于快速访问 Web。
- Oracle Management Agent：安装在被管理的数据库服务器上，用于监控被管理数据库的运行状态、资源使用、信息变更等，将这些信息传送到 OMS，OMS 同时将这些数据存入 Repository。

25.2　OEM的安装

OEM 的安装十分简单，不需要下载额外的软件，它集成在数据库的 RDBMS 软件中，在安装数据库软件以及创建数据库的过程中，只要点选相应的选项即可自动完成 OEM 的安装。本节我们介绍几种方式，以利于读者有选择地使用。

25.2.1　第一种安装方式

在安装数据库软件的时候，如果选取了 Create and configure a database 选项，这样在创建数据库时会自动安装 OEM 软件。图 25-2 是安装 Oracle 数据库软件 11g R2 时的界面。

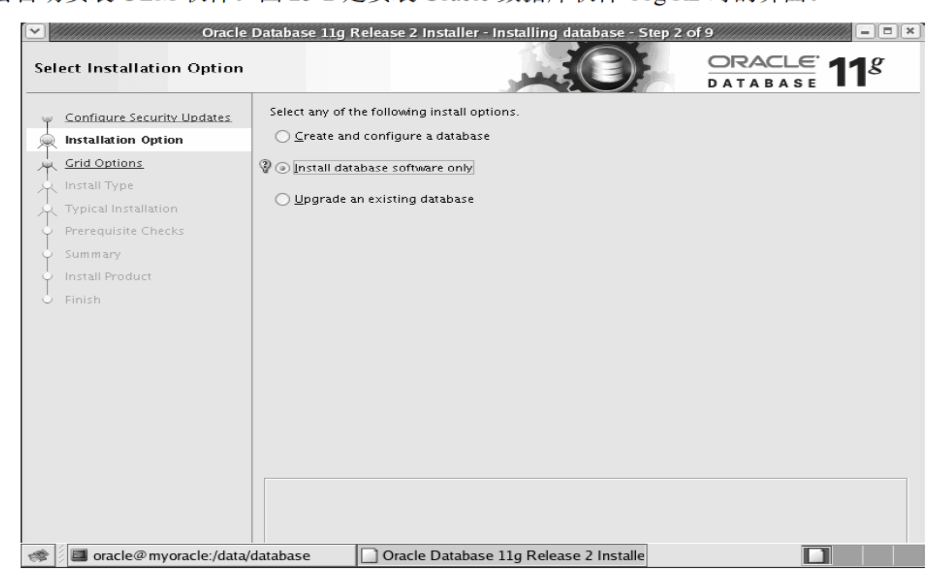


图 25-2　安装 Oracle 数据库 RDBMS

在图 25-2 中，要求在 3 个安装选项中选择安装方式，图中选择的是 Instan database software only，要自动安装 OEM 来管理数据库，此时可以选择第一项 Create and configure a database，此后的 OEM 安装过程不需要用户干预，自动完成。在数据库软件安装完毕并创建完数据库之后，会出现如下提示，告诉我们 OEM 的相关信息，如图 25-3 所示。

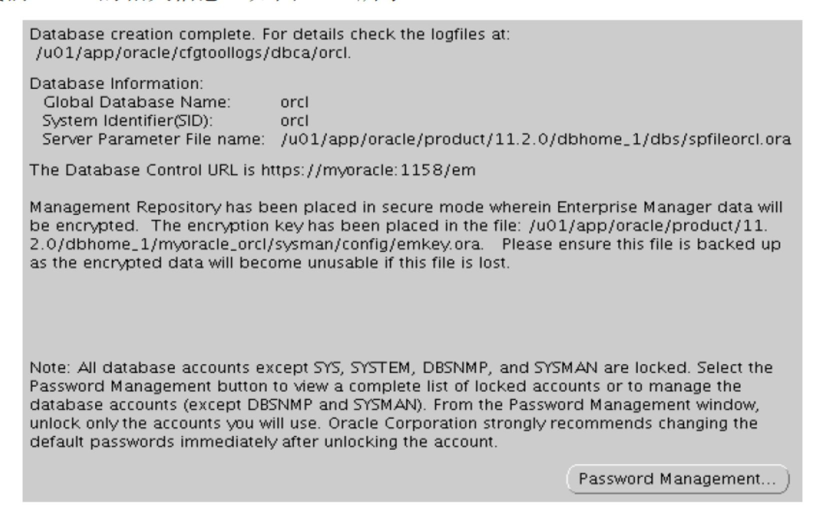


图 25-3　EM 的相关信息

图 25-3 中的重点是访问 OEM 的 URL 信息 https://myoracle:1158/em。此处的 1158 是端口号，必须输入后面的/em。

25.2.2　第二种安装方式

使用 DBCA 的 Configure Database Options 选项配置 OEM，下面我们先启动 DBCA，如图 25-4 所示。



图 25-4　启动 DBCA 界面

接着单击“下一步”按钮，如图 25-5 所示。



图 25-5　选择“配置数据库选件”

在图 25-5 中，需要选择“配置数据库选件”，然后单击“下一步”此时服务器上安装的数据库会自动被搜索到，选择需要安装 OEM 的数据库，如图 25-6 所示。



图 25-6　选择对应的数据库

接着单击“下一步”会出现 OEM 相关选项，如图 25-7 所示。



图 25-7　安装 OEM 选项

这种方式十分简单，全部由 DBCA 工具完成，如果 DBCA 不能使用，也可以使用 EMCA 指令来创建，这就是第三种安装方式。

25.2.3　第三种安装方式

EMCA 指令设置与 EM 相关的一系列操作，如设置新的或重新创建 Repository 存储区、删除 Repository 存储区、配置 OEM Database Control 以及删除 OEM Database Control 等操作。下面我们先看两个指令，emca –repos recreate 与 emca –config dbcontrol db。前者用于重新创建 Repository 存储区，后者用户创建 OEM Database Control。我们结合这两个指令来重新创建 OEM（如果没有安装 OEM 则需要设置新的 Repository 存储区即可，将 emca –repos recreate 更换为 emca –repos create 即可）。

下面是重建 OEM 的例子。

例子 25-1　重建 OEM

```
    [oracle@myoracle ~]$ emca -config dbcontrol db -repos  recreate

    STARTED EMCA at Mar 9, 2013 11:17:54 AM
    EM Configuration Assistant, Version 11.2.0.0.2 Production
    Copyright (c) 2003, 2005, Oracle.  All rights reserved.

    Enter the following information:
    Database SID: orcl                //要求输入 SID
    Database Control is already configured for the database orcl
    You have chosen to configure Database Control for managing the database orcl
    This will remove the existing configuration and the default settings and perform
a fresh configuration
    Do you wish to continue? [yes(Y)/no(N)]: yes  //确定是否继续
    Listener ORACLE HOME [ /u01/app/oracle/product/11.2.0/dbhome 1 ]:
    Password for SYS user:          //要求输入 SYS 用户密码
    Password for DBSNMP user:     //要求输入 DBSNMP 用户密码
    Password for SYSMAN user:     //要求输入 SYSMAN 用户密码
    Password for SYSMAN user: Email address for notifications (optional):
```

```
    Outgoing Mail (SMTP) server for notifications (optional):
    -----------------------------------------------------------------

    You have specified the following settings  //以下内容是 EMCA 学习到的

    Database ORACLE HOME ............ /u01/app/oracle/product/11.2.0/dbhome 1

    Local hostname ....... myoracle
    Listener ORACLE HOME .............. /u01/app/oracle/product/11.2.0/dbhome 1
    Listener port number ................ 1521
    Database SID ................ orcl
    Email address for notifications ...............
    Outgoing Mail (SMTP) server for notifications ...............

    -----------------------------------------------------------------
    Do you wish to continue? [yes(Y)/no(N)]: yes  //确认后继续删除并安装 OEM
    Mar 9, 2013 11:18:47 AM oracle.sysman.emcp.EMConfig perform
    INFO:       This       operation       is       being       logged       at
/u01/app/oracle/cfgtoollogs/emca/orcl/em ca 2013 03 09 11 17 54.log.
    Mar 9, 2013 11:18:49 AM oracle.sysman.emcp.util.DBControlUtil stopOMS
    INFO: Stopping Database Control (this may take a while) ...
    Mar 9, 2013 11:18:59 AM oracle.sysman.emcp.EMReposConfig invoke
    INFO: Dropping the EM repository (this may take a while) ...
    Mar 9, 2013 11:24:54 AM oracle.sysman.emcp.EMReposConfig invoke
    INFO: Repository successfully dropped
    Mar 9, 2013 11:24:55 AM oracle.sysman.emcp.EMReposConfig createRepository
    INFO: Creating the EM repository (this may take a while) ...
    Mar 9, 2013 11:39:20 AM oracle.sysman.emcp.EMReposConfig invoke
    INFO: Repository successfully created
    Mar     9,     2013     11:39:39     AM     oracle.sysman.emcp.EMReposConfig
uploadConfigDataToRepository
    INFO: Uploading configuration data to EM repository (this may take a while) ...
    Mar 9, 2013 11:41:39 AM oracle.sysman.emcp.EMReposConfig invoke
    INFO: Uploaded configuration data successfully
    Mar     9,     2013     11:41:56     AM     oracle.sysman.emcp.util.DBControlUtil
configureSoftwareLib
    INFO: Software library configured successfully.
    Mar     9,     2013     11:41:56     AM     oracle.sysman.emcp.EMDBPostConfig
configureSoftwareLibrary
    INFO: Deploying Provisioning archives ...
    Mar     9,     2013     11:42:52     AM     oracle.sysman.emcp.EMDBPostConfig
configureSoftwareLibrary
    INFO: Provisioning archives deployed successfully.
    Mar 9, 2013 11:42:52 AM oracle.sysman.emcp.util.DBControlUtil secureDBConsole
    INFO: Securing Database Control (this may take a while) ...
    Mar 9, 2013 11:43:08 AM oracle.sysman.emcp.util.DBControlUtil secureDBConsole
    INFO: Database Control secured successfully.
    Mar 9, 2013 11:43:08 AM oracle.sysman.emcp.util.DBControlUtil startOMS
    INFO: Starting Database Control (this may take a while) ...
    Mar 9, 2013 11:44:45 AM oracle.sysman.emcp.EMDBPostConfig performConfiguration
    INFO: Database Control started successfully
    Mar 9, 2013 11:44:45 AM oracle.sysman.emcp.EMDBPostConfig performConfiguration
    INFO:  >>>>>>>>>>>  The  Database  Control  URL  is  https://myoracle:1158/em
<<<<<<<<<<<
    Mar 9, 2013 11:44:51 AM oracle.sysman.emcp.EMDBPostConfig invoke
    WARNING:
    ************************  WARNING  ************************

    Management Repository has been placed in secure mode wherein Enterprise Manager
data  will  be  encrypted.   The  encryption  key  has  been  placed  in  the  file:
```

```
/u01/app/oracle/product/11.2.0/dbhome_1/myoracle_orcl/sysman/config/emkey.ora.
Please ensure this file is backed up as the encrypted data will become unusable
if this file is lost.

    ***********************************************************
    Enterprise Manager configuration completed successfully
    FINISHED EMCA at Mar 9, 2013 11:44:51 AM
```

下面是 EMCA 的常用指令：

- Emca –config dbcontrol db ： 配置 OEM Database Control。
- Emca –deconfig dbcontrol db ： 删除 OEM Database Control。
- Emca –repos create ： 创建新的 Repository 存储区。
- Emca –repos drop: 删除 Repository 存储区。
- Emca –repos recreate ： 重建 Repository 存储区。

读者可以自己独立执行这些指令，观察其作用，同时注意这些指令可以组合使用，如例子 25-1 中删除并重建 OEM 的指令。

25.3 OEM的启动与关闭

当使用 OEM 时，必须保证 Database Control 已经启动，指令 EMCTL 就是用于管理 OEM Database Control 的启动关闭以及查看状态信息等。

例子 25-2 查看 Oracle Enterprise Manager 的状态

```
    Emctl status dbconsole
    [oracle@myoracle ~]$ emctl status dbconsole
    Oracle Enterprise Manager 11g Database Control Release 11.2.0.1.0
    Copyright (c) 1996, 2009 Oracle Corporation.  All rights reserved.
    https://myoracle:1158/em/console/aboutApplication
    Oracle Enterprise Manager 11g is not running.
    ------------------------------------------------------------------
    Logs are generated in directory /u01/app/oracle/product/11.2.0/dbhome_1/
myoracle_orcl/sysman/log
```

在例子 25-2 中，我们查看了 Oracle Enterprise Manager 11g 的状态，显然它并没有运行，此时如果尝试登录 OEM 服务器会报错。下面启动 Oracle Enterprise Manager。

例子 25-3 启动 Oracle Enterprise Manager

```
    Emctl start dbconsole
    [oracle@myoracle ~]$ emctl start dbconsole
    Oracle Enterprise Manager 11g Database Control Release 11.2.0.1.0
    Copyright (c) 1996, 2009 Oracle Corporation.  All rights reserved.
    https://myoracle:1158/em/console/aboutApplication
    Starting Oracle Enterprise Manager 11g Database Control .......... started.
    ------------------------------------------------------------------
    Logs are generated in directory /u01/app/oracle/product/11.2.0/dbhome_1/
myoracle_orcl/sysman/log
```

例子 25-4 关闭 Oracle Enterprise Manager

```
    Emctl stop dbconsole
```

```
    [oracle@myoracle ~]$ emctl stop dbconsole
    Oracle Enterprise Manager 11g Database Control Release 11.2.0.1.0
    Copyright (c) 1996, 2009 Oracle Corporation.  All rights reserved.
    https://myoracle:1158/em/console/aboutApplication
    Stopping Oracle Enterprise Manager 11g Database Control ...
     ...  Stopped.
```

可以通过 emca –h 指令查看 emca 的其他功能。

```
    [oracle@myoracle ~]$ emctl -h
    Oracle Enterprise Manager 11g Database Control Release 11.2.0.1.0
    Copyright (c) 1996, 2013 Oracle Corporation.  All rights reserved.
       Oracle Enterprise Manager 1g Database Control commands:
           emctl start | stop dbconsole
           emctl status | secure | setpasswd dbconsole
           emctl  config  dbconsole  -heap size  <size value>  -max perm size
<size value>
          emctl status agent
              ………….
```

下面使用 emctl status agent 查看 Agent 的状态。只有 Agent 正常运行才能获取数据库服务器的管理和状态信息。

例子 25-5　查看 Agent 的状态

```
    [oracle@myoracle ~]$ emctl status agent
    Oracle Enterprise Manager 11g Database Control Release 11.2.0.1.0
    Copyright (c) 1996, 2013 Oracle Corporation.  All rights reserved.
    ---------------------------------------------------------------
    Agent Version:      10.2.0.4.2
    OMS Version:        10.2.0.4.2
    Protocol Version:   10.2.0.4.2
    Agent Home:         /u01/app/oracle/product/11.2.0/dbhome 1/myoracle orcl
    Agent binaries:     /u01/app/oracle/product/11.2.0/dbhome 1
    Agent Process ID:   13252
    Parent Process ID:  13233
    Agent URL:      https://myoracle:3938/emd/main
    Repository URL: https://myoracle:1158/em/upload/
    Started at:     2013-03-11 09:53:42
    Started by user:    oracle
    Last Reload:        2013-03-11 09:53:42
    Last successful upload:                     2013-03-11 10:15:16
    Total Megabytes of XML files uploaded so far:  0.56
    Number of XML files pending upload:         0
    Size of XML files pending upload(MB):       0.00
    Available disk space on upload filesystem:  22.32%
    Data channel upload directory          : /u01/app/oracle/product/11.2.0/
dbhome 1/myoracle orcl/sysman/recv
    Last successful heartbeat to OMS:           2013-03-11 10:15:50
    ---------------------------------------------------------------
    Agent is Running and Ready
```

例子 25-5 中说明 Agent 是否运行以及 Reposiroty 是否可用。最后的输出说明 Agent 正在运行。其他指令读者可以自行测试。

25.4　OEM监控数据库运行

在启动 OEM 后，我们就可以使用 EM 管理、监控和维护数据库了。下面首先登录 Web 服务器，如图 25-8 所示输入地址 https://myoracle:1158/em，此时会出现如图 25-8 所示的界面。要求我们输入必需的信息：用户名、密码以及连接数据库时所使用的角色。笔者输入 sys 用户名，密码为 Oracle1234，使用 SYSDBA 角色登录。

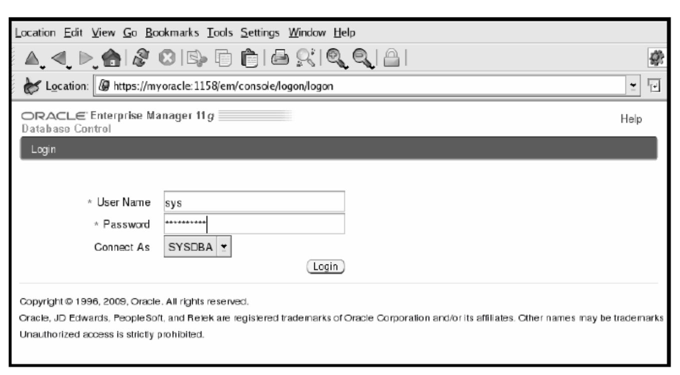


图 25-8　OEM 登录界面

Oracle 的 OEM 预设的用户有 3 个：SYS、SYSTEM 和 SYSMAN。对于前两个是 Oracle 的高级管理员用户，而 SYSMAN 是专门用于 OEM 的用户。

OEM 用户的管理权限分为两种：管理员权限和非管理员权限。前者拥有很高的权限，如监控、修改设置、数据库审计、升级以及备份恢复等，而非管理员用户只能查看信息而不能修改任何设置。

单击图 25-8 中的 Login，即可看到 OEM 的主功能界面，如图 25-9 所示，其中提供了 7 个标签供用户选择。分别是 Home、Performace、Avaliability、Server、Schema、Data Movement、Software and Support。我们从这 7 个方面分别介绍其功能，这里我们不会面面俱到，只希望读者了解基本功能，需要时可以到相应的标签中去找，因为是图形界面，所以用户在掌握数据库基本概念和基本操作技能的基础上会很快学会使用。

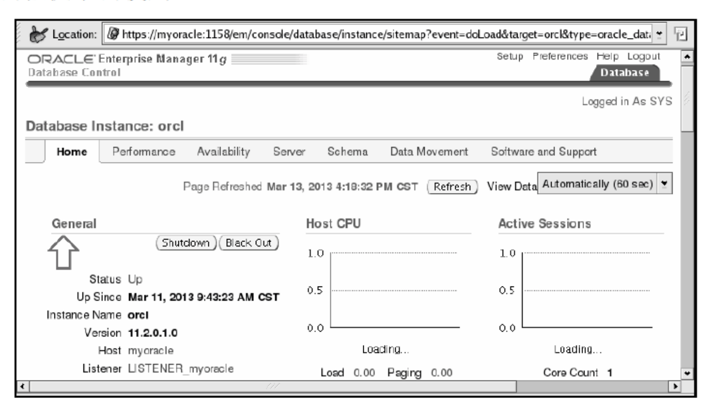


图 25-9　OEM 首页信息

- Home：显示整个系统当前状况，如 CPU、活跃会话统计、SQL 相应时间统计等。
- Performance：显示数据库的性能信息。
- Available：提供数据库的备份恢复设置、以及安全备份等。
- Server：涉及数据库服务器的多个管理领域，包括存储管理、数据库配置管理、调度管理、统计信息管理、资源管理以及用户管理。
- Schema：模式管理的作用对象是构成模式的对象，它的功能包括数据库对象管理、程序管理、物化视图管理、用户定义类型以及 XML 数据库管理等。
- Data Movement：包括数据的逻辑备份（EXPDP、IMPDP）、迁移数据库文件、流管理、高级复制管理。
- Software and Support：软件和支持部分获得软件的状态信息，如获得 HOST 的配置信息、数据库补丁需求情况等。

25.4.1　Home 目录

如图 25-10 所示，在 Home 目录中呈现了系统的整体状态信息，这个信息通过图的形式呈现，十分直观，而且具体的指标和内容可以通过单击相应内容获得，如想获得 CPU 被数据库使用的情况就可以单击 Host CPU 下的 orcl。其他情况类似。

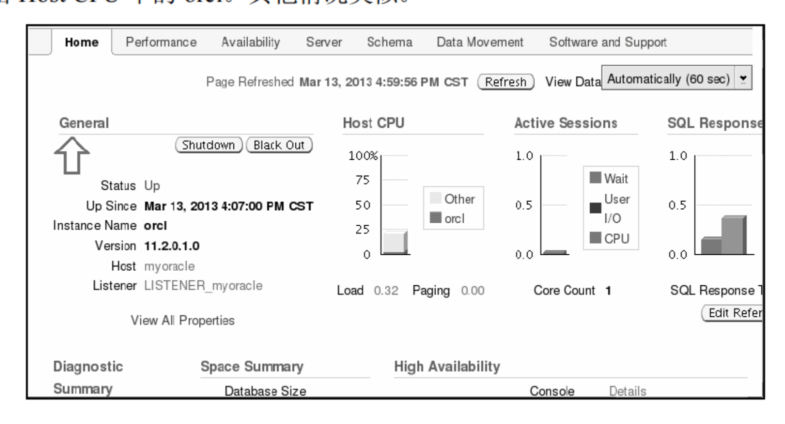


图 25-10　Home 目录主页

在 General 部分呈现了数据库状态以及监听信息，首先数据库的状态 Status 是 Up，为启动状态，Up Since 为启动开始时间，Instance Name 是数据库实例名，Version 为数据库版本，Host 是数据库所在的主机，Listener 是为该数据库服务的监听器。我们可以通过单击 Listener 的名称 LISTENER_myoracle 来查看监听器的具体信息，如图 25-11 所示。

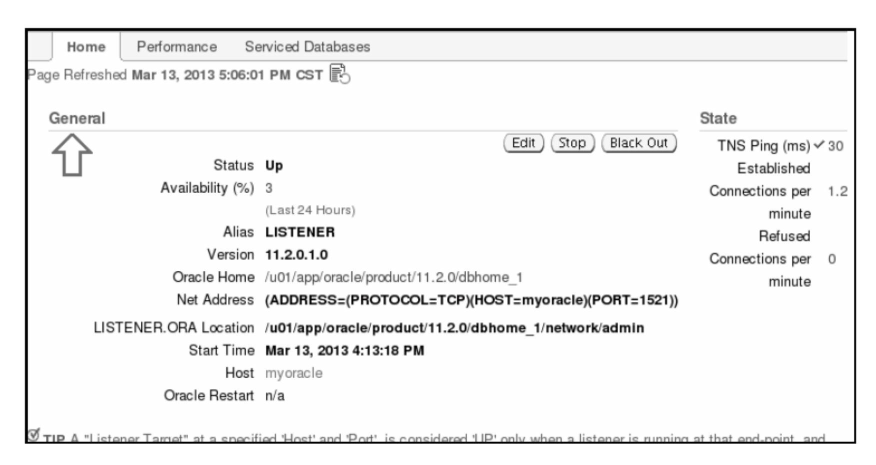


图 25-11　监听器的详细信息

图 25-11 是监听器的主页部分，还有两个功能即 Performance 与 Serviced Databases。如需要则可以继续选择相应的 Tab，如 Serviced Databases 查看该监听器服务的数据库有哪些，如图 25-12 所示。

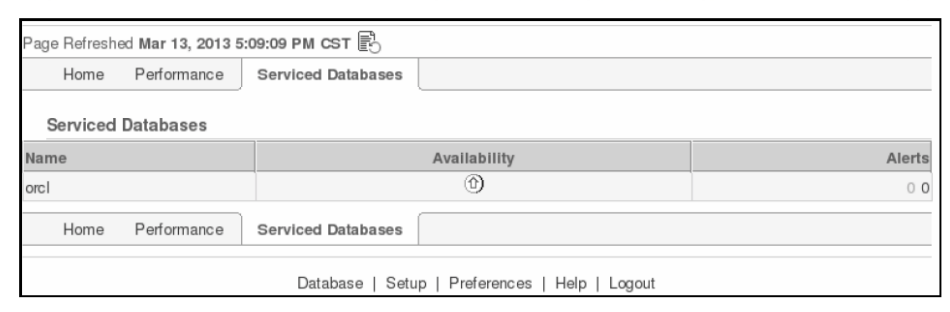


图 25-12　监听器的 Serviced Databases

25.4.2　Performance 部分

这个部分主要提供了主机监控、用户连接监控以及 I/O 性能，还有其他相关的连接选项用于辅助判断。

- 主机监控：这部分监控是否存储在 CPU 瓶颈。呈现了 CPU 使用率的曲线图，清晰地给出 CPU 使用率的分布。如果在某个时间段内 CPU 使用率极高，已经造成了系统性能问题，如终端用户反映应用系统变慢，此时就需要分析用户会话，或者更确切地说是哪些 SQL 正在消耗 CPU 资源。
- 平均活跃会话监控：这部分是分析实例性能问题的关键部分，它呈现了实例中的等待时间。某个时间段内会话正在等待的事件，哪些会话正在等待那些事件，这些事件使用不同的颜色表示。如磁盘 I/O、网络通信、调度、等待 CPU 等。如果存在大量等待而影响了系统性能，此时则需要结核使用 TOP ACTIVITY 分析 TOP SQL 和 TOP SESSOIN，继而使用 SQL 优化顾问提供优化 SQL 的效果。

另外还有额外监控链接，这些链接对于分析数据库性能是十分有帮助的，如图 25-13 所示。

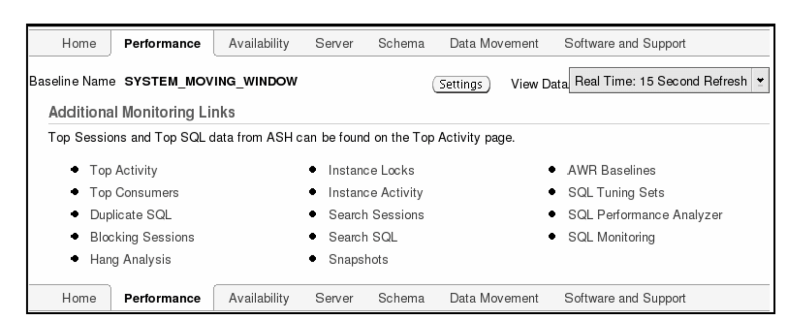


图 25-13　Performance 中额外监控的链接

监控当前实例的活跃状况（Instance Activity），如图 25-14 所示。

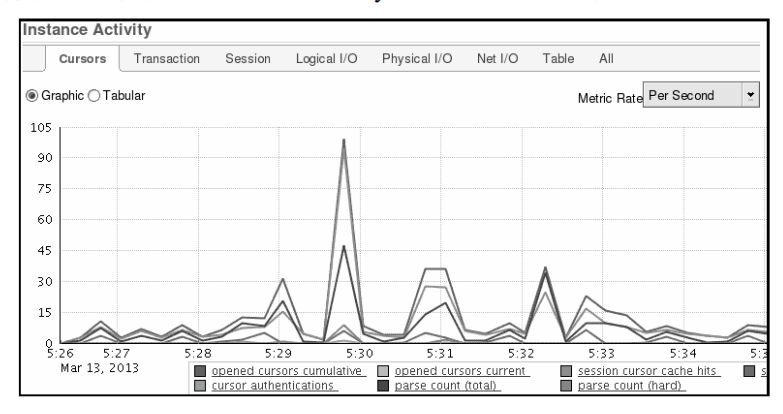


图 25-14　实例状态信息监控主页面

图 25-14 显示的内容每 15 秒钟刷新一次，也可以使用 Tabular 通过表格的方式显示。其中给出了累计打开的游标数、当前打开的游标数、硬解析数量依据总的解析数。每一个 Tab 都有图形化 Graphic 和表格式 Tabular 数据，具体内容读者可以自行查看。

25.4.3　Availablity 部分

这部分包括备份恢复设置、备份恢复管理以及安全备份。如图 25-15 所示，我们重点介绍备份恢复设置、备份恢复方法。

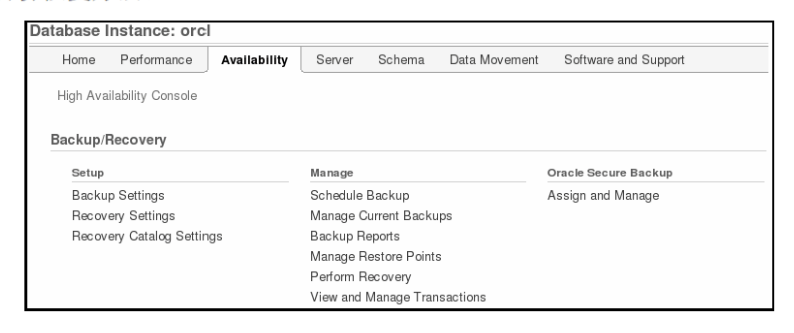


图 25-15　Availability 主页

在 Setup 中，我们进行 Backup Setting 设置，这是备份设置。如图 25-16 所示，给出了具体需要的备份设置。

Backup Settings
Device Backup Set Policy
Disk Settings
Test Disk Backup
Parallelism 1
Concurrent streams to disk drives
Disk Backup Location
The flash recovery area is the current disk backup location. If you would like to override the disk backup location, specify an existing directory or diskgroup.
Disk Backup Type ◉ Backup Set
An Oracle backup file format that allows for more efficient backups by interleaving multiple backup files into one output file.
○ Compressed Backup Set
An Oracle backup set in which the data is compressed to reduce its size.
○ Image Copy
A bit-by-bit copy of database files that can be used as-is to perform recovery.
Tape Settings
Tape drives must be mounted before performing a backup. You should verify that the tape settings are valid by clicking on 'Test Tape Backup', before saving them.
Test Tape Backup
Clear Tape Configuration
Tape Drives
Concurrent streams to tape drives
Tape Backup Type ◉ Backup Set
An Oracle backup file format that allows for more efficient backups by interleaving multiple backup files into one output file.
○ Compressed Backup Set
An Oracle backup set in which the data is compressed to reduce its size.
Oracle Secure Backup Domain
Oracle Secure Backup Domain
Version on Database Server Unknown
Oracle Secure Backup Domain Target Not Defined
Backup Storage Selectors Configure
A backup storage selector is recommended when backing up the database to tape.
Media Management Settings
If you need to configure a media manager from a third-party vendor, specify the library parameters.
Media Management Vendor Library Parameters
Host Credentials
To save the backup settings, supply operating system login credentials to access the target database.
* Username sys
* Password ******
☑ Save as Preferred Credential

图 25-16 备份设置

- 磁盘设置部分。设置并行度以及备份目录，设置磁盘备份的类型，包括备份集、压缩备份集或者映像备份。
- 磁带设置部分。在磁带使用之前必须保证磁带已经挂接，这里可以测试是否挂接。这里的备份类型只有两种即备份集和压缩备份集，显然不能使用映像备份，因为映像备份是数据块复制。

- 介质管理设置部分。如果我们使用了第三方的产品，此时需要配置介质管理器，指定连接该产品的库参数。这些参数由生产商的产品提供。
- 主机设置部分。需要提供操作系统的用户名和密码，这个用户是安装数据库的用户，可以访问目标数据库。

恢复设置需要配置几部分，包括实例恢复、介质恢复与闪回恢复 3 个部分。下面我们按照顺序依次介绍这些配置项。

- 实例恢复，如图 25-17 所示。

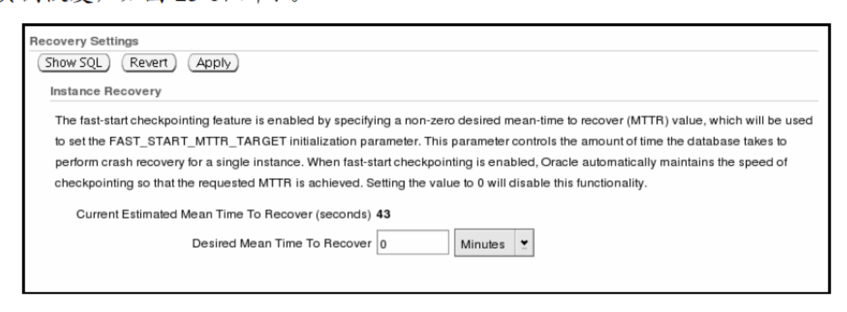


图 25-17　实例恢复设置

实例恢复不需要用户干预，当数据库重新启动时后台进程会自动检测，通过 REDO 记录实现实例恢复。此时考虑到实例恢复的时间要求，一旦如图 25-17 所示设置了平均恢复时间，则会按照时间间隔发生检验点事件，这样在检验点之前的事务都已经提交，不再需要实例恢复，只有检验点之后的事务需要实例恢复，这样就减少了实例恢复的时间，这个设置本质上是修改参数 FAST_START_MTTR_TARGET 的值。

- 介质恢复，如图 25-18 所示。

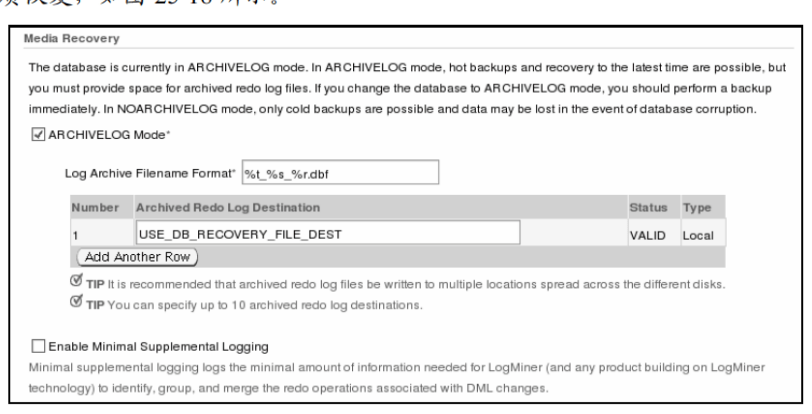


图 25-18　介质恢复设置

介质恢复的本质是通过备份以及归档日志实现数据库的完全恢复，所以需要设置归档日志的信息，这里需要填写恢复数据库时归档日志的文件格式以及归档目录地址。最后选择是否启动最小的补充日志。所有基于 LogMinner 技术的产品会需要这个补充日志识别、分组或者聚合与 DML 操作相关的 Redo 操作。

- 闪回恢复，如图 25-19 所示。

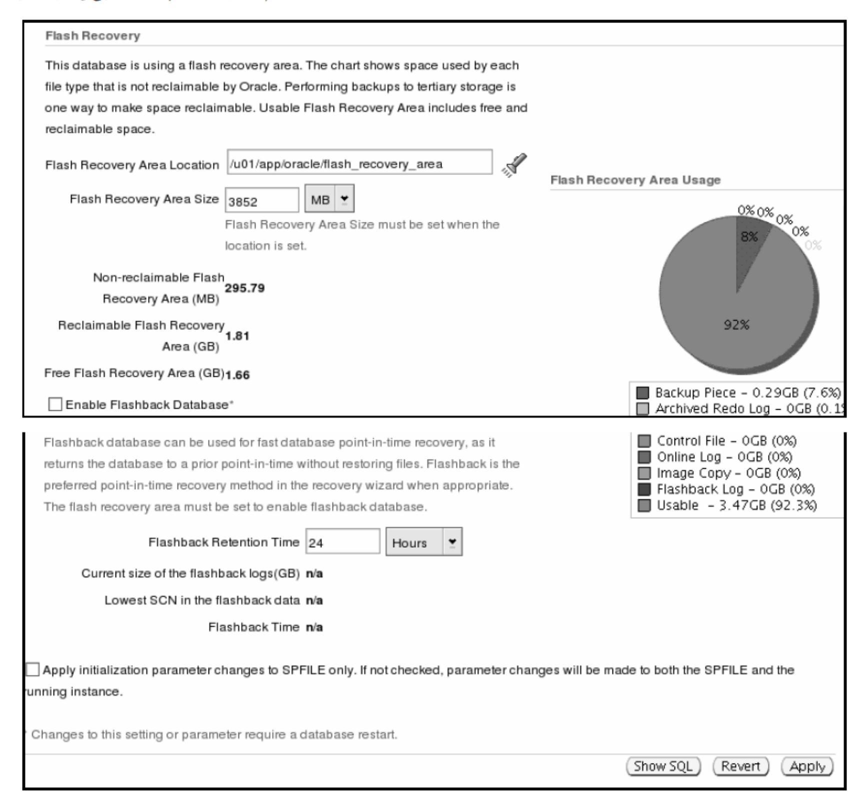
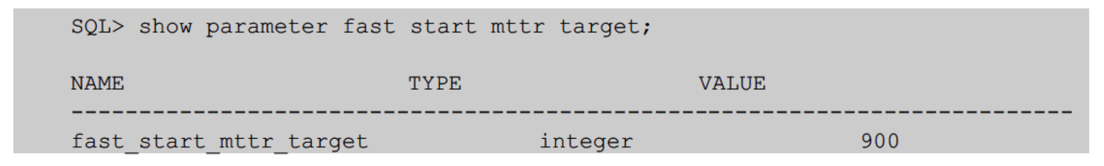


图 25-19　闪回恢复设置

闪回恢复需要设置两个参数一个闪回恢复区目录，一个是闪回恢复区大小，这两个参数分别为：db_recovery_file_dest 和 db_recovery_file_dest_size。最后选择是否启动闪回数据库。一旦启动闪回数据库则需要设置闪回到过去的时间长度。Oracle 将尽可能保证恢复到这个时间长度之内的任意时间点。这要看闪回恢复区的大小是否收到限制。

在修改完恢复设置参数后，单击 Apply 按钮，应用修改。然后我们可以通过指令检查修改是否成功，如例子 25-6 所示。

例子 25-6　查看实例恢复时间参数设置



```
SQL> show parameter fast start mttr target;

NAME                              TYPE                      VALUE
---------------------------------------------------------------------------
fast_start_mttr_target                    integer                    900
```

显然，此时该参数不是默认的 0，而是 900，通过换算我们知道这里的单位应该是秒，因为我们设置该参数的值为 15 分钟，而这个参数的数值为 900。

- 管理当前备份。

当前备份管理包括两个部分，一个是备份集部分，一个是映像备份部分，如图 25-20 所示。

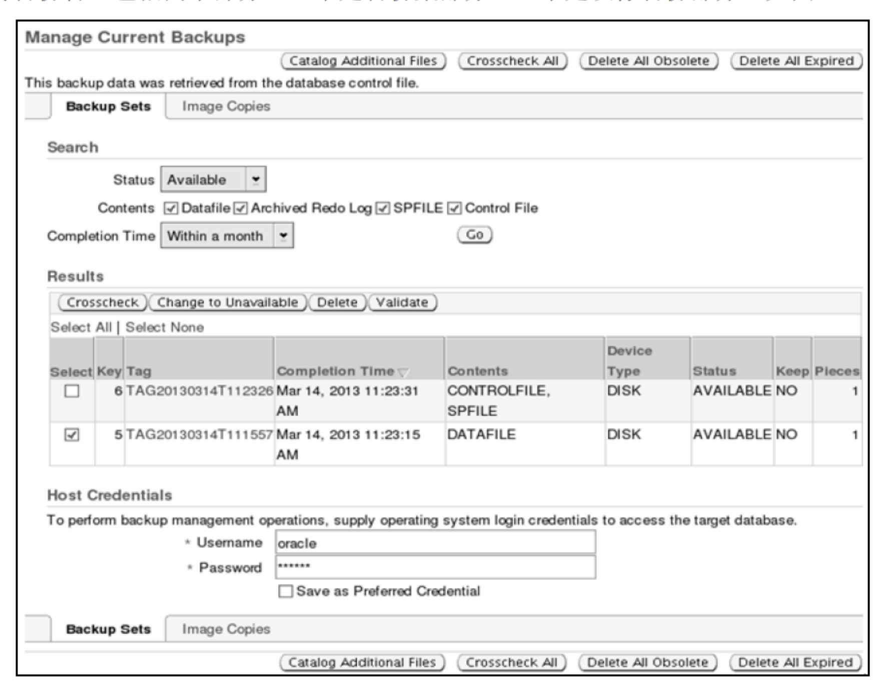


图 25-20 管理备份（备份集部分）

当我们已经备份数据库后，使用备份管理可以很容易实现数据文件的交叉检验、有效性检验，以及删除过期或者不满足备份保留条件的备份。如在图 25-20 中，我们可以选择对数据文件进行交叉检验（作用）。此时需要填写访问该数据文件的操作系统的用户和密码。单击 Result 旁边的 Crosscheck，如图 25-21 所示。



图 25-21 交叉检验所有备份的数据文件

本质上这个操作是执行 RMAN 的脚本指令 CROSSCHECK BACKUPSET 5。显然这个指令我们完全可以在 RMAN 中操作。但是这里要更简单，操作更友好。在单击 Yes 后会进行数据文件检验，最后给出检验结果，这个操作在后台进行，如果此时浏览器关闭不影响对数据文件的 Crosscheck 检验。当校验成功完成，则自动回到管理备份主界面，即图 25-20，但是会有一行提示信息。

```
Information
     The Crosscheck operation has been successfully executed.
```

当然也可以对数据文件进行 Validate 有效性检验，此时会默认提交一个 Job，该 Job 的程序也是执行一个 RMAN 脚本，读者可以自行测试。

- 执行恢复操作，如图 25-22 所示。

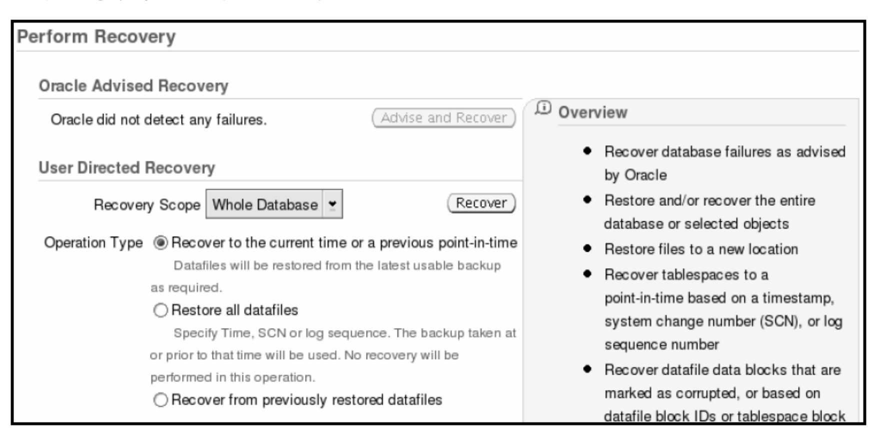


图 25-22　选择恢复操作

在这个对话框中，需要首先选择恢复范围（Scope），包括整个数据库、数据文件、表空间、归档日志、表以及事务。对应不同的 Scope 有不同的操作类型，图 25-22 为对于整个数据库的恢复。如果选择对数据文件恢复，则如图 25-23 所示。

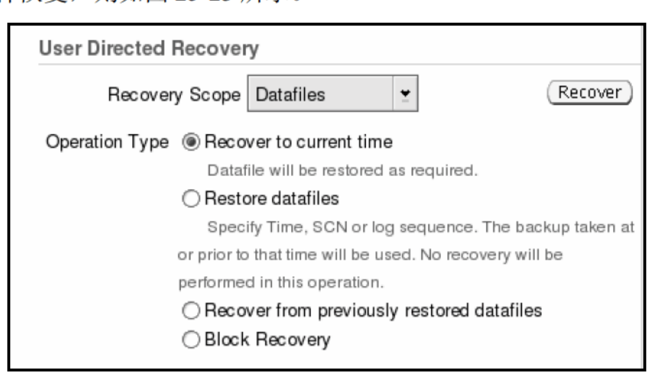


图 25-23　对数据文件恢复

对于数据文件恢复，可以恢复到当前时间点、复原数据文件以及实现数据块恢复。

25.4.4　Server 部分

这部分内容比较多，主要涉及数据库服务器的管理，包括存储管理、数据库配置管理、调度管理、统计管理、资源管理、安全管理、查询优化以及变更数据库。下面我们依次介绍这些内容的具体管理项目。

- 存储管理。存储管理包括控制文件管理、表空间管理、临时表空间管理、数据文件管理、回滚段管理、重做日志组管理、归档日志管理、迁移到 ASM、改变表空间为本地管理。

图 25-24 是数据文件管理的内容。

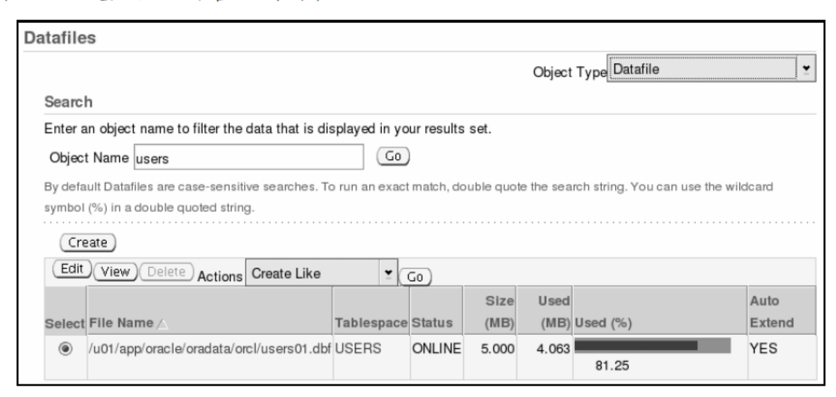


图 25-24　Server 部分的数据文件管理

在数据文件管理中，在 Object Name 中输入需要查看的数据文件的表空间名称会过滤不相关的数据文件信息，增加显示的准确性。可以编辑、查看、创建、删除数据文件。通过这个管理窗口，DBA 可以很清楚地了解数据文件的状态、大小、使用比例等，更直观地掌握数据文件的状态。

- 数据库配置管理。这部分包括了内存顾问、自动 UNDO 管理、初始化参数管理以及查看数据库特性。如图 25-25 所示为初始化参数管理部分。

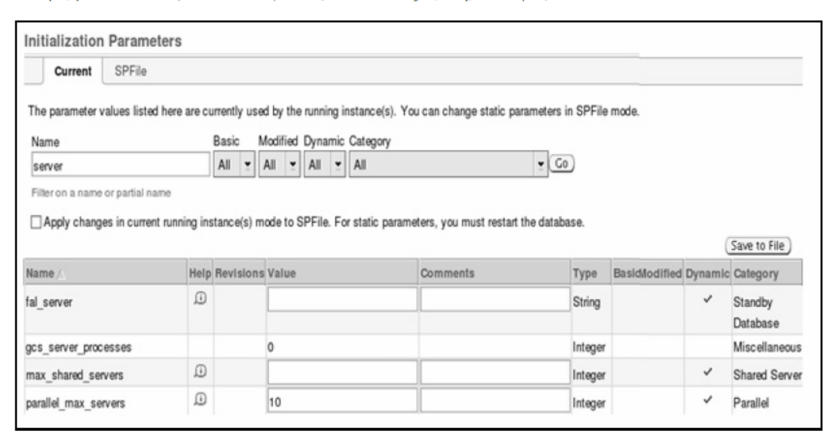


图 25-25　初始化参数管理部分

初始化参数管理实现对当前的初始化参数的查看和修改。在 Name 字段输入相应参数所含的字符可以过滤出所有相关的参数。如图 25-25 中查询和 server 相关的参数。当前的 max_shared_servers 参数没有设置，可以在这里修改这个参数，在 Value 的字段输入值，单击窗口最下面的 Apply 按钮，因为这是一个动态参数，所以可以直接修改。

- 调度管理。这部分包括 Job 管理、Chains 管理、调度器、Program 管理、Job 类管理、窗口以及窗口组管理等。调度是 Oracle 允许数据库自身，或者用户完成指定的管理任务而设置的组件。调度结合了调度时间（或窗口）和调度任务（Program 或 PL/SQL 程序）。下面我们查看当前的数据库的全部 Jobs 信息，如图 25-26 所示。



图 25-26　调度管理

当前窗口显示了所有的调度任务，通过输出知道调度的名称、所属的用户、被调度时间，调度所属于的类以及调度执行的次数。同时可以实现对选择的调度的管理，如查看调度定义、编辑调度内容、删除调度、立即运行该调度以及创建新的调度任务。

对于其他 Server 部分的其他内容不再赘述，需要强调的是变更数据库管理（Change Database）是对于集群环境而言，只有在 RAC 环境下才出现删除与增加实例的内容。对于管理集群的 Grid Control 中，这部分可以实现其功能。

25.4.5　Schema 部分

这部分包括数据库对象管理、程序管理、物化视图管理、数据字典变更管理、用户定义数据类型管理、XML 数据库、工作空间管理器以及文本管理器。使用频率比较高的是数据库对象管理、程序管理以及物化视图管理，这些是 DBA 管理数据库时使用频率较高的部分。如图 25-27 所示为索引管理部分。

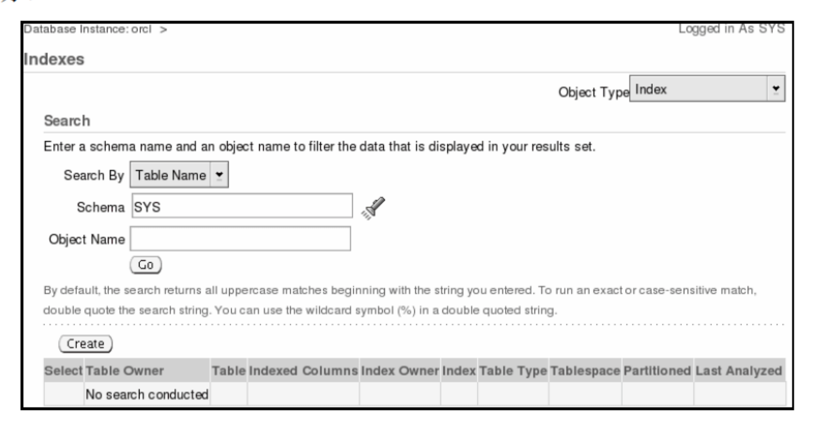


图 25-27　索引管理

通过索引管理，可以查看某个用户，或者某个表上所拥有或建立的索引，可以查看某个索引的定义，编辑并删除索引，当然也可以增加索引。

图 25-28 为程序管理部分创建存储过程的窗口。

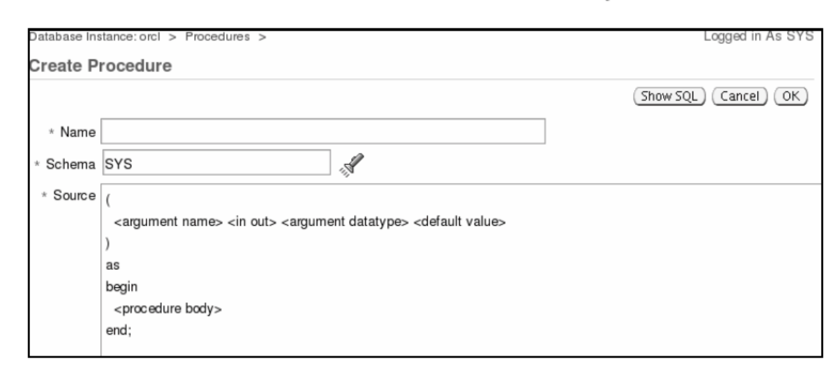


图 25-28　创建存储过程

在这个窗口中，DBA 可以针对某个模式，创建该模式所拥有的存储过程，这个过程可以被应用程序调用，也可以被 Job 调度使用。

25.4.6　Data Movement 部分

数据迁移部分包括移动行数据、移动数据库文件、Streams 流管理以及高级复制，如图 25-29 所示。

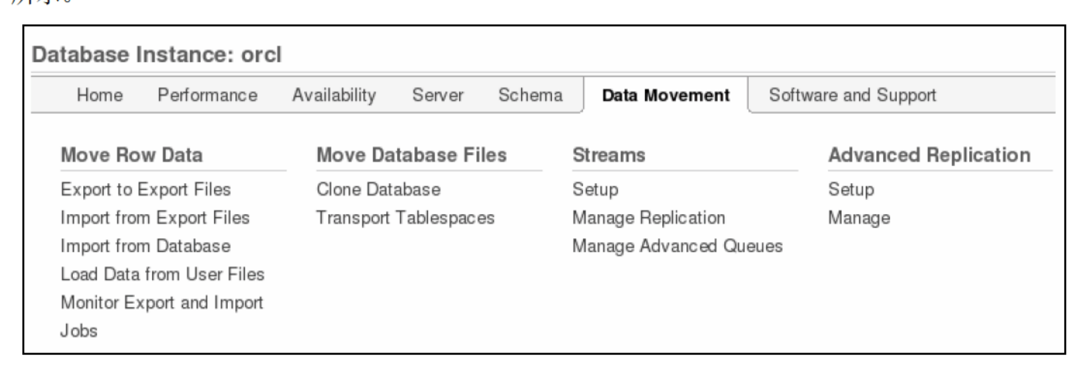


图 25-29　数据迁移部分

其中移动数据库文件包含复制数据库以及迁移表空间，对于迁移表空间我们完全可以通过 EXPDP 和 IMPDP 的方式手工实现，显然使用图形化工具简化了 DBA 的操作。

而使用 EXPORT 导出为 EXPORT 文件的方式，我们也完全可以通过 EXPORT 工具实现，这里图形化界面简化了我们的操作，Oracle 内部是通过一个 Job 实现该操作，并将输出文件存放在某个默认目录下。这些在图像化操作中都会有体现。读者可以自行体验。

25.4.7　Software and Support 部分

这部分提供了丰富的软件管理内容，如图 25-30 所示。包括软件的配置管理，如主机的配置以及状态信息、资源使用率信息等。数据库软件补丁管理，可以查看当前的补丁建议，以及按照补丁的前提要求，并实施打补丁任务。

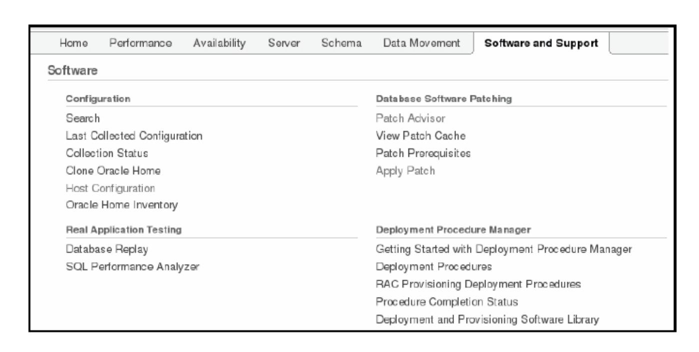


图 25-30　Software and Support 部分

Database Replay 部分可以捕捉生成库的工作负载，并测试工作负载的变化对数据库的影响，从而为生成库的调整给出科学的评估。

Procedure 管理器用于管理当前已经部署好的 Procedure，管理内容包括编辑、查看、运行以及监控部署好的 Procedure。而 Procedure 管理器要正常工作，需要启动一个监视 Daemon Job。它用来监视当前正在运行的 Procedure。可以使用 pafctl 指令启动、关闭和查看该 Job。Pafctl 指令位于 OMS ORACLE_HOME>/bin 目录下。

启动、关闭和查询 Provisioning Daemon 的指令分别为 start、stop 和 status。具体为启动指令 pafctl start、关闭指令为 pafctl stop、查看状态为 pafctl status。其中启动该 job 时要求输入 Job 的启动间隔，该数值以分钟为单位。

SYSMAN 用户密码是指定连接 repository 的用户 SYSMAN 的密码。

INTERVAL 是运行 Provisioning Daemon Job 的时间间隔，单位为分钟。如果没有设置，默认为 3 分钟。

在上面的描述中我们知道，在启动、关闭以及查看 Daemon Job 时需要输入 sysman 密码，这个用户用于管理 OEM。然后输入该 Job 的执行间隔。下面是具体启动 Daemon Job 的例子。

例子 25-7　启动 Procedure 管理器所需要的 Daemon Job

```
[oracle@myoracle ~]$ pafctl
Usage : pafctl start|stop|status <sysman password> <interval>
[oracle@myoracle ~]$ pafctl start
Enter repository user password :
Enter interval [default 3]: 3
Provisioning Daemon is Up, Interval = 3
```

下面是软件配置部分的具体示例，我们查看主机配置信息，即使用 EM 同时可以监控主机资源的使用，如主机的 CPU、内存的使用状态等。

在下午 5 点 09 分笔者查看了主机配置中的 Performance 窗口，发现此时的 CPU 以及内存是比较平缓的，因为除了操作系统几乎没有消耗这些资源的操作。随后笔者执行了一个 SQL 语句，该语句为 select * from dba_segments。显然执行该 SQL 语句时，需要 SQL 语句硬解析，需要 CPU 资源，同时需要读取磁盘上的数据到 database cache。需要消耗内存资源，当然共享池也会增加该 SQL 语句的父游标和子游标，占有一点内存，而此时的从共享池分配 chunk 存储游标也需要 CPU 资源，所以在执行该 SQL 语句后，EM 监控到的 CPU、内存以及磁盘 I/O 都有变化，如图 25-31 所示。

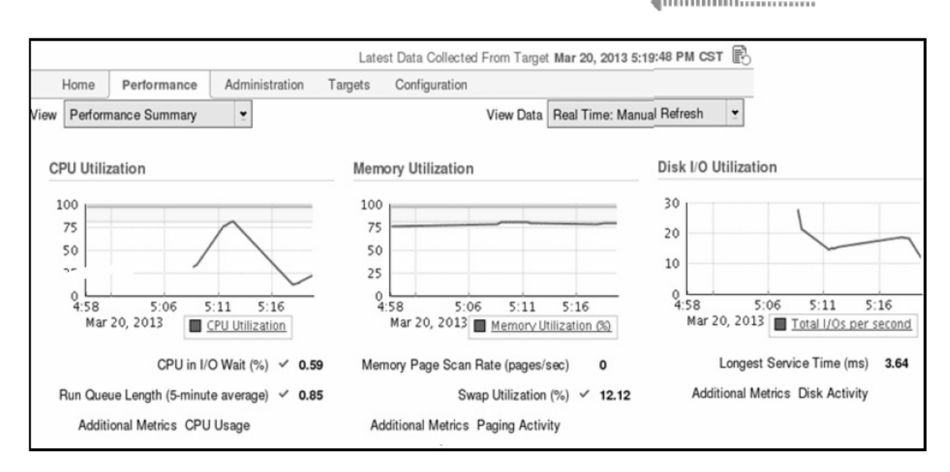


图 25-31　主机的 Performance 状态

在图 25-31 中，我们可以清楚地看到 SQL 语句执行过程中，不同资源的消耗情况，如在 5:11 分左右 CPU 的使用率开始上升，而此时的磁盘 I/O 却处于局域内的最低值，因为此时更多的是在执行硬解析，耗费 CPU 资源多。之后 CPU 使用率达到一个峰值，因为此时共享池管理以及磁盘 I/O 操作都需要 CPU 调度管理。在 5 点 18 分左右 SQL 语句执行结束，此时 CPU 使用率最低，而磁盘 I/O 达到峰值。由于数据量不是很大，所以内存使用率在 SQL 语句执行期间略微有点上升。

显然通过 EM 我们可以更清楚地观察 CPU、内存以及磁盘 I/O 的使用率，从而更好地了解我们所维护的数据库系统的主机资源状态。

25.5　本章小结

本章介绍了 EM 的架构、几种安装方式以及基本的管理，如关闭、启动 EM 控制台。而后介绍通过 OEM 监控与管理数据库的各种功能概述。使用 EM 管理数据库本质上还是使用 SQL 语句管理数据库，以及通过对 Oracle 的各种数据字典访问获得数据库的状态、性能信息。只是使用 EM 更加直观快捷，省去了 DBA 大量的时间，如在监控主机资源方面，不论是什么操作系统平台，使用 EM 却可以省去 DBA 通过各种操作系统指令查看资源状态的过程，而且是实时刷新，可以更好地满足 DBA 的需求。

第 26 章

Oracle数据库实例优化

在理解了 Oracle 数据库的架构、各组件作用以及基本的管理任务后，就可以进一步通过优化来深入理解各组件的作用了。本章我们介绍 Oracle 数据库实例优化，Oracle 实例由内存组件和相关的后台进程组成，这些内存组件提高了数据库系统的运行，而后台进程负责管理系统或者内存组件，使得整个数据库系统协调一致地工作，但是 Oracle 数据库的实例并不满足所有的生产数据库系统的需求，即使使用自动内存管理，Oracle 也只是提出一个建议。本章我们讨论在 Oracle 实例优化中最典型的 SGA 组件优化方法。

26.1 详解SGA与实例优化

Oracle 的 SGA 是指系统全局区，它是数据库运行期间使用的一段公有内存，即所有使用数据库的用户都可以访问这部分内存，它包括共享池、重做日志缓冲区、数据库缓存高速缓冲区、Java 池、大池以及流池。

因为优化 SGA 就是调整这些数据库组件的参数，这些组件是实例优化的操作对象，从而提高系统的运行效率，如提高用户查询的响应时间等。图 26-1 是 SGA 的组成示意图。

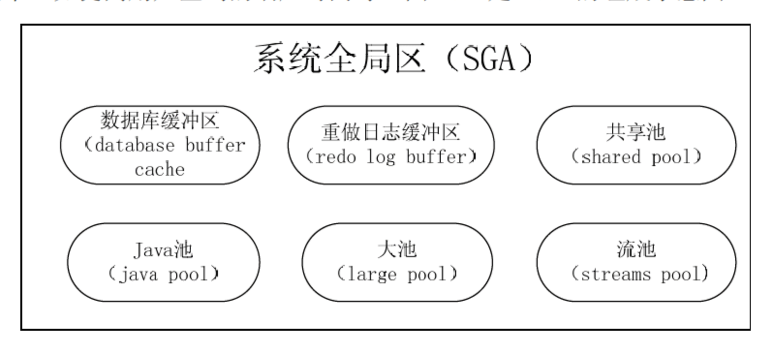


图 26-1 Oracle 的 SGA 组成图

这里我们解释图 26-1 中各个组件的作用以及涉及的参数，这样读者在修改上述组件时就更有针对性，做到“有的放矢”。

- 数据库缓冲区：该区域存放用户从数据库中读取的数据，在用户查找数据库时会首先在数据库缓存中搜索，如果没有才会读数据库文件。所以该区域不能设置得过小，不然频繁的

读取数据文件会增大查询时间，因为磁盘 I/O 是耗时的行为。

- 重做日志缓冲区：该缓冲区放置用户改变的数据，所有变化了的数据和回滚需要的数据都暂时保存在重做日志缓冲区中。涉及的参数为 log_buffer。如下所示我们查询重做日志缓冲区的大小。

例子 26-1　查询重做日志缓冲区的大小

```
SQL> show parameter log buffer;

NAME                                  TYPE        VALUE
-------------------------------- -------------------- -----------------
log_buffer                                integer      260242640
```

- 共享池：共享池包括数据字典高速缓存和库高速缓存，库高速缓存存放 Oralce 解析的 SQL 语句、PL/SQL 过程、包以及各种控制结构如锁、库缓冲句柄等。而数据字典高速缓存保存执行 sql 语句所需的各种数据字典定义、如表和列的定义、用户访问表的权限等。
- Java 池：执行 java 代码的区域。它为 Oracle 数据库中运行的 JVM（Java 虚拟机）分配的一段固定大小的内存。
- 大池：该内存区提供大型的内存分配，在共享服务器连接模式下提供会话区，在使用 RMAN 备份时也使用该内存区作为磁盘 I/O 的数据缓冲区。
- 流池：该区域称为流内存，是为 Oracle 流专用的内存池，流是 Oracle 数据库中的一个数据共享，其大小可以通过参数 streams_pool_size 动态调整。

在 Oracle11g 以及更高版本中，SGA 的中内存参数可以动态修改，但是总的内存大小受到参数 SGA_MAX_SIZE 的限制。在安装数据库时，这个参数的值是默认的，而实际的生产数据库往往需要重新设置一个新值，以利用操作系统中充足的内存资源。我们查看参数 SGA_MAX_SIZE 的值，如例子 26-2 所示。

例子 26-2　查看参数 SGA_MAX_SIZE 的值

```
SQL> show parameter sga_max_size;

NAME                                  TYPE        VALUE
----------------------------- ------------------- ------------
sga_max_size                             big integer     576M
```

我们继续查看 SGA 信息，如例子 26-3 所示，查看当前数据库的 SGA 信息。

例子 26-3　查看 SGA 信息

```
SQL> show sga;

Total System Global Area  603979776 bytes
Fixed Size                  170380 bytes
Variable Size             222301108 bytes
Database Buffers          373293056 bytes
Redo Buffers                7135232 bytes
```

上述输出的第一个参数 Total System Global Area 其实和例子 26-2 中查到的是一个数据，二者

大小相等，如下所示。

```
SQL> select 576*1024*1024 bytes
  2  from dual;

    BYTES
-----------------
 26039797726
```

下面我们调整 SGA 的最大尺寸，目的是增大 Oracle 在整个内存中所占的比例，但是不能太大，一般可以设置为当前内存大小的一半即可。我们修改参数 SGA_MAX_SIZE 以修改 SGA 的尺寸。如例子 26-4 所示。

例子 26-4　修改 SGA_MAX_SIZE 参数

```
SQL> alter system set sga_max_size= 700M scope = spfile;

系统已更改。
```

例子 26-4 中，我们把 SGA_MAX_SIZE 改为了 700M，下面我们查询这次修改，如例子 26-5 所示。

例子 26-5　查询参数 SGA_MAX_SIZE 修改结果

```
SQL> show parameter sga max size;

NAME                                TYPE       VALUE
------------------------------------------------ -------------------- -------
sga_max_size                                big integer   576M
```

观察 VALUE 的值发现该值为 576MB，没有修改，其实需要向读者说明参数 SGA_MAX_SIZE 是静态参数，需要重启数据库后方可生效。我们先不关闭数据库，继续对 SGA 的优化。

我们查看在 Oracle 的静态参数中有哪些和 SGA 相关，如例子 26-6 所示。

例子 26-6　查看和 SGA 相关的静态参数

```
SQL> show parameter sga;

NAME                                TYPE       VALUE
--------------------------------------- -------------------- -------------
lock sga                             boolean         FALSE
pre page sga                          boolean         FALSE
sga_max_size                          big integer       576M
sga_target                             big integer       576M
```

下面我们依次介绍参数 LOCK_SGA、PRE_PAGE_SGA 和 SGA_TARGET，看这些参数对于优化 SGA 有什么作用。

- LOCK_SGA 的含义及优化

该参数的作用是将 SGA 锁定（lock）在物理内存内，这样就不会发生 SGA 使用虚拟内存的情况，显然这样可以提高数据的读取速度，记住磁盘 I/O 操作永远是尽量避免或减少的。该参数的默

认值为 FALSE，即不将 SGA 锁定在内存中。下面，我们修改参数 LOCK_SGA 为 TRUE，如例子 26-7 所示。

例子 26-7　设置参数 LOCK_SGA 为 TRUE

```
SQL> alter system set lock sga = true scope = spfile;

系统已更改。
```

该参数是静态参数，需要重启数据库才可生效。

- PRE_PAGE_SGA 的含义及优化

该参数的作用是启动数据库实例时，将整个 SGA 读入物理内存，对于内存充足的系统而言，这样显然可以提高系统运行效率。我们修改该参数为 TRUE，如例子 26-8 所示。

例子 26-8　设置参数 PRE_PAGE_SGA 为 TRUE

```
SQL> alter system set pre page sga= true scope = spfile;

系统已更改。
```

下面我们关闭数据库并重启数据库，如例子 26-9 所示。

例子 26-9　关闭并重启数据库

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup
ORA-32004: obsolete and/or deprecated parameter(s) specified
ORACLE 例程已经启动。

Total System Global Area  734003200 bytes
Fixed Size                 171172 bytes
Variable Size            356518044 bytes
Database Buffers          369098752 bytes
Redo Buffers               7135232 bytes
数据库装载完毕。
数据库已经打开。
```

现在，我们查看刚才修改的 3 个参数：SGA_MAX_SIZE、LOCK_SGA、PRE_PAGE_SGA，如例子 26-10 所示。

例子 26-10　查看与 SGA 相关的参数修改结果

```
SQL> show parameter sga;

NAME                                 TYPE        VALUE
------------------------------------ ---------------------- --------
lock_sga                              boolean          TRUE
```

```
pre page sga                          boolean         TRUE
sga_max_size                          big integer      576M
sga_target                             big integer      576M
```

从上述输出可以看出参数 LOCK_SGA 和 PRE_PAGE_SGA 的值都为 TRUE，参数 SGA_MAX_SIZE 的值也修改为 700MB。读者或许会问，参数 SGA_TARGET 是什么作用呢？

- SGA_TARGET 的含义及优化

在 Oracle10g 11g 以及以上的版本中，提供了内存的自动管理，在 Oracle 11g 中更是实现了 SGA 和 PGA 的自动管理，只要设置 memory_target 与 memory_max_size 两个参数即可，这样 Oracle 可以根据业务需要和服务器自身的软硬件环境自动调整一些内存参数。如果不设置这两个参数，也可以设置 SGA 和 PGA 各自自动管理，兼容 10g 中的内存管理方式。参数 SGA_TARGET 决定是否使用 SGA 自动管理，该参数的默认值和系统的 SGA_MAX_SIZE 一样大，当该参数值不为 0 时，则启动 SGA 的自动管理，该参数可以动态修改，下面我们修改该参数的值为 700MB，如例子 26-11 所示。

例子 26-11　修改参数 SGA_TARGET 的值

```
SQL> alter system set sga target = 700M;

系统已更改。
```

读者可以自行查看修改结果，这里不再给出查询结果，

既然 SGA 可以自动管理，但不是所有的内存组件都可以自动调整，那么那些 SGA 的内存是否可以自动调整呢，下面是 SGA 可以自动调整的内存组件。

- 共享池。
- Java 池。
- 大池。
- 数据库缓冲区。
- 流池。

这些组件的尺寸不需要用户干预，其值自动设置为 0。我们使用视图 v$parameter 查看这些自动调整的内存组件的信息。

例子 26-12　查看自动调整的内存组件的信息

```
SQL> col name for a30
SQL> col value for a20
SQL>select name,value,isdefault
  2  from v$parameter
  3  where name in ('shared_pool_size','large_pool_size',
  4* 'java_pool_size')

NAME                          VALUE                ISDEFAULT
-------------------------------------------------------------------------
shared_pool_size                     0                    TRUE
large_pool_size                      0                    TRUE
```

```
java_pool_size                         0                    TRUE
```

虽然这些参数是可以自动调整的，但是用户依然可以使用 ALTER SYSTEM SET 指令修改内存组件的尺寸，如下例所示，修改 java_pool_size 的值为 10MB。

```
SQL> alter system set java_pool_size = 10MB;

系统已更改。
```

下面我们查询修改结果，如例子 26-13 所示。

例子 26-13　查询参数 java_pool_size 的修改结果

```
SQL> show parameter java pool size;

NAME                                 TYPE        VALUE
------------------------------------------------------------- ----------
java_pool_size                           big integer     12M
```

显然参数 java_pool_size 的值不再是 0，而 VALUE 值变为 12M。读者或许会问前面设置该参数值为 10M，怎么变成了 12M 呢，其实 Oracle 会针对系统自身情况做一些调整。这是数据库自己的行为，读者不必太在意。

26.2　将程序常驻内存

在 Oracle 数据库中有一个软件包 DBMS_SHARED_POOL，它提供过程 KEEP 和 UNKEEP，将用户经常使用的程序如存储过程、触发器、序列号、游标，以及 JAVA SOURCE 等数据库对象长期保存在一个内存结构中，这个内存区就是共享池（shared pool）。对于用户频繁使用的这些数据库对象而言，将其常驻内存可以减少磁盘 I/O 从而减少用户的响应时间。本节我们先讲解几个数据块缓冲池，分别解释他们的作用以及使用时机，然后介绍如何将一个存储过程常驻内存，最后介绍创建软件包 DBMS_SHARED_POOL 的 dbmspool.sql 过程，从而更清楚地理解软件包中各种过程的作用以及参数含义。

26.2.1　创建软件包 DBMS_SHARED_POOL

在 Oracle 数据库中，软件包 DBMS_SHARED_POOL 不是默认安装的，所以需要执行一个.sql 脚本文件来创建该软件包。它有两个经常使用的过程：KEEP 和 UNKEEP。KEEP 过程实现将程序常驻内存，而 UNKEEP 将指定的程序清除出内存。如果读者开始就尝试执行软件包 DBMS_SHARED_POOL 的 KEEP 过程，则会提示错误。

首先使用 DBA 用户登录数据库。

例子 26-14　登录数据库并执行 KEEP 过程

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL> execute dbms_shared_pool.keep('HR.SECURE_DML');
BEGIN dbms_shared_pool.keep('HR.SECURE_DML'); END;
```

```
     *
第 1 行出现错误:
ORA-06550: 第 1 行, 第 7 列:
PLS-00201: 必须声明标识符 'DBMS SHARED POOL.KEEP'
ORA-026550: 第 1 行, 第 7 列:
PL/SQL: Statement ignored
```

显然，提示执行失败，这说明没有可用的软件包，需要手工创建该软件包，如例子 26-15 所示。该软件包在笔者的电脑上位于目录 F:\app\Administrator\product\11.1.0\db_1\RDBMS\ADMIN 下，脚本文件名为 dbmspool.sql。其实在这个目录下还有很多其他脚本文件，如我们熟悉的创建 SCOTT 用户的脚本文件 scott.sql。

例子 26-15 创建 DBMS_SHARED_POOL 软件包

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL>@F:\app\Administrator\product\11.1.0\db_1\RDBMS\ADMIN\dbmspool.sql;

程序包已创建。


授权成功。


视图已创建。


程序包体已创建。
```

从输出可以看出，此时成功创建软件包，并且在执行脚本文件 dbmspool.sql 的过程中，实现了授权和视图创建，并同时创建了过程 KEEP 和 UNKEEP，（当然还有其他过程）我们接下来会进行更详细的介绍。

如果用户在 SCOTT 用户或其他非 SYSTEM 用户下登录数据库并且尝试创建软件包 DBMS_SHARED_POOL，会提示出错，如下所示。

```
SQL> connect scott/tiger@orcl
已连接。
SQL> F:\app\Administrator\product\11.1.0\db_1\RDBMS\ADMIN\dbmspool.sql;

程序包已创建。


授权成功。

  from dba object size
      *
第 4 行出现错误:
ORA-01031: 权限不足
警告: 创建的包体带有编译错误。
```

显然从输出可以看出当前用户缺少足够的权限，只要使用 SYSTEM 用户登录且赋予 DBA 角色即可。

在成功创建软件包 DBMS_SHARED_POOL 后就可以使用它的过程 KEEP 来将程序常驻内存了。

26.2.2　将程序常驻内存的过程

软件包 DBMS_SHARED_POOL 是过程的集合，包含常用的 KEEP 和 UNKEEP 过程，使用 KEEP 过程将用户频繁使用的程序常驻共享池中，使用 UNKEEP 将指定的程序从共享池中清除。我们先选择并查看用户 HR 的一个过程。

首先使用 SYSTEM 用户登录数据库。

```
SQL> connect system/oracle@orcl
已连接。
```

然后，通过数据字典 DBA_OBJECTS 查询用户 HR 的一个存储过程，我们假设该存储过程是用户程序频繁调用的过程，然后将其常驻内存，如例子 26-16 所示，先查找用户 HR 的一个存储过程。

例子 26-16　查看用户 HR 拥有的存储过程

```
SQL>  select object name,object type
  2   from dba objects
  3   where object_type ='PROCEDURE'
  4   and owner ='HR';

OBJECT NAME          OBJECT TYPE
---------------------------------- ------------------------------
SECURE_DML           PROCEDURE
ADD_JOB_HISTORY       PROCEDURE
```

从查找结果可以看出，用户 HR 有两个存储过程，一个为 SECURE_DML，一个为 ADD_JOB_HISTORY。我们将过程 SECURE_DML 常驻内存，或许读者想知道如何查看该过程的内容，毕竟对过程的功能了解得越多就越能理解为什么将其常驻内存，为此 Oralce 提供了数据字典 DBA_SOURCE(USER_SOURCE)。

例子 26-17　查看过程 SECURE_DML 的内容

```
SQL> SET LINE 100
SQL>select line,text
  2  from dba source
  3* where name ='SECURE DML'

     LINE TEXT
---------- --------------------------------------------------------------
        1 PROCEDURE secure dml
        2 IS
        3 BEGIN
        4   IF TO_CHAR (SYSDATE, 'HH24:MI') NOT BETWEEN '08:00' AND '18:00'
        5        OR TO CHAR (SYSDATE, 'DY') IN ('SAT', 'SUN') THEN
        6      RAISE APPLICATION ERROR (-20205,
        7             'You may only make changes during normal office hours');
```

```
          8   END IF;
          9 END secure_dml;

已选择 9 行。
```

该过程的作用很简单，就是判断某种状态下的系统时间如果不在 8 点到 18 点之间或者日期为周六或周日就提示错误'You may only make changes during normal office hours'。好了，我们不过多分析这个过程，读者只需要知道数据字典 dba_source 的作用即可。下面演示如何将过程 secure_dml 常驻内存，如例子 26-18 所示。

例子 26-18 将过程 secure_dml 常驻内存

```
SQL> EXECUTE dbms_shared_pool.keep('HR.SECURE_DML');

PL/SQL 过程已成功完成。
```

输出提示已经成功创建 PL/SQL 过程，为了确认创建结果，我们使用数据字典 V$db_object_cache。它的作用是存储关于数据库对象在缓存中的信息。

例子 26-19 查看用户 HR 的存储过程是否保存在共享池中

```
SQL> col name for a20
SQL> select name,namespace,sharable_mem,executions,kept
  2  from v$db object cache
  3* where owner ='HR'

NAME           NAMESPACE       SHARABLE_MEM EXECUTIONS  KEP
-------------------- ---------------------- ---------- ---------- -------
SECURE_DML   TABLE/PROCEDURE    1737         0           YES
```

此时，输出说明用户 HR 的数据库对象即存储过程 SECURE_DML，已经保存在共享池中，因为 KEPT 列的值为 YES。

既然可以使得一个程序常驻内存，同样有方法将其从内存清除，现在我们使用软件包 DBMS_SHARED_POOL 的 UNKEEP 过程将用户 HR 的过程 SECURE_DML 清除出内存。

例子 26-20 将用户 HR 的过程 SECURE_DML 清除出内存

```
SQL> EXECUTE dbms_shared_pool.unkeep('HR.SECURE_DML');

PL/SQL 过程已成功完成。
```

在执行清除任务后，我们再次使用数据字典 v$db_object_cache 来查看清除结果，如例子 26-21 所示。

例子 26-21 查看是否从内存清除过程 SECURE_DML

```
SQL>  select name,namespace,sharable_mem,executions,kept
  2   from v$db_object_cache
  3   where owner ='HR';

NAME           NAMESPACE        SHARABLE MEM EXECUTIONS KEP
-------------------- ------------------------ ------------------ --------
SECURE_DML  TABLE/PROCEDURE  1737            0           NO
```



如果将实现了常驻内存的程序从内存清除，该程序的记录仍然在数据字典 v$db_object_cache 中，只是 KEPT 列的值为 NO。如果从没有将一个程序常驻内存，则在数据字典 v$db_object_cache 中不存在该程序的任何记录。

26.2.3　从 DBMSPOOL 脚本理解软件包 DBMS_SHARED_POOL

上面我们使用软件包 DMBS_SHARED_POOL 将用户 HR 的一个过程 SECURE_DML 常驻内存，同时又使用了软件包过程 UNKEEP 将该过程从内存清除。那么软件包 DBMS_SHARED_POOL 到底是如何创建的呢？我们来分析脚本文件 dbmspool.sql，从而可以更清楚地理解软件包的作用，以及其中包含的过程的含义。以下从脚本文件中截取部分内容详细说明。

```
create or replace package dbms_shared_pool is
  ------------
  --  OVERVIEW
  --
  --  This package provides access to the shared pool.  This is the
  --  shared memory area where cursors and PL/SQL objects are stored.
```

这部分说明该软件包的作用是提供对共享池的访问，使得游标或者 PL/SQL 对象（如存储过程、函数等）可以存储在共享池中。

该软件包中还包括 4 个函数，我们先介绍 KEEP 函数和 UNKEEP 函数。

关于 KEEP 函数脚本中的内容如下所示。

```
procedure keep(name varchar2, flag char DEFAULT 'P');
  --  Keep an object in the shared pool.  Once an object has been keeped in
  --    the shared pool, it is not subject to aging out of the pool.  This
  --    may be useful for certain semi-frequently used large objects since
  --    when large objects are brought into the shared pool, a larger
  --    number of other objects (much more than the size of the object
  --    being brought in, may need to be aged out in order to create a
  --    contiguous area large enough.
  --    WARNING:  This procedure may not be supported in the future when
  --    and if automatic mechanisms are implemented to make this
  --    unnecessary.
```

该函数的作用就是将一个数据库对象常驻共享池，使得频繁访问的数据库对象如大对象等提高用户的访问响应时间，提高访问速度。该函数有两个参数。

- 第一个参数 name。

该参数用来说明数据库对象的名字，这些数据库对象可以是 PL/SQL 过程、触发器、序列号，以及 JAVA 对象。如果是 PL/SQL 过程可以使用“模式名.过程名”的方式指定特定模式的数据库过程名，如 scott.hispackage。

- 第二个参数 flag。

该参数的作用是说明要常驻的数据库对象的类型，默认类型为包、过程或函数。否则需要使

用一个字符变量说明对象类型。字符值与其代表的对象类型的对应关系如下所示。

```
Value        Kind of Object to keep
-----        ----------------------
P          package/procedure/function
 Q          sequence
  R          trigger
  T          type
  JS         java source
  JC         java class
  JR         java resource
  JD         java shared data
 C          cursor
```

关于 UNKEEP 过程的内容如下所示。

```
procedure unkeep(name varchar2, flag char DEFAULT 'P');
  -- Unkeep the named object.
  --   WARNING:  This procedure may not be supported in the future when
  --   and if automatic mechanisms are implemented to make this
  --   unnecessary.
  -- Input arguments:
  --   name
  --     The name of the object to unkeep.  See description of the name
  --     object for the 'keep' procedure.
  --flag
  --     See description of the flag parameter for the 'keep' procedure.
  -- Exceptions:
  --   An exception will raised if the named object cannot be found.
```

因为有了对于 KEEP 过程的详细说明，而过程 UNKEEP 的参数含义和 KEEP 过程的相同，所以不再重述 UNKEEP 过程的参数含义。

在该脚本文件中还有一条重要的授权语句，如下所示。

```
grant execute on dbms_shared_pool to execute_catalog_role
```

将对软件包 DBMS_SHARED_POOL 的执行权利赋予角色 EXECUTE_CATALOG_ROLE，而我们当前的 SYSTEM 用户具有 DBA 权限所以自动具有角色 EXECUTE_CATALOG_ROLE，也可以通过数据字典查询当前用户具有的角色，如例子 26-22 所示。

例子 26-22　查看当前用户的角色信息

```
SQL> select *
  2  from dba roles
  3  where role like 'EXECUTE%';

ROLE                           PASSWORD
---------------------------------------------- -------------------------
EXECUTE_CATALOG_ROLE           NO
```

例子 26-22 说明了角色 EXECUTE_CATALOG_ROLE 的存在，所以当前用户在创建了软件包 DBMS_SHARED_POOL 后就可以使用它了。

其实，笔者是希望读者在使用脚本文件时一定要仔细阅读脚本文件的内容，这样就可以从本

质上理解一个软件包的作用和其中包含的其他过程。

26.3　将数据常驻内存

在生产数据库中，为了提高用户的访问速度，对于经常使用的表，可以使其常驻内存中，避免了对该表访问时产生频繁的磁盘 I/O 行为，这样可以提高用户的访问响应时间，虽然造成一定的内存占用，但是使用内存访问确实提高了访问的响应时间，在某种程度上是有效的。而当不需要频繁访问该表时，DBA 可以将其从内存中清除。本节我们再次学习 Oralce 的各种数据块的缓存池，通过分析了解将数据常驻内存的必要性和可行性。然后给出一个具体的例子将 SCOTT 用户的 EMP 表和一个索引常驻内存。

26.3.1　再论数据块缓存池

在 Oracle 数据库体系结构的介绍中，读者已经知道在数据库块写到磁盘文件之前，或者从磁盘文件读取数据之后，首先需要将数据块缓存在数据库高速缓存中，所以需要适当设置该缓冲区的大小以满足用户需求。在 Oracle 8 之后的版本中，用户可以把 SGA 中段的已缓存块放在 3 个缓冲池中。

- 默认池（default pool）：所有的段都放在这个池中，即原先的缓冲区池，如果没有指定数据的缓存位置，默认将数据缓存在这个池中。
- 保持池（keep pool）：对于用户频繁访问的数据（如表或索引等数据库对象的数据块）可以放置在这个候选的缓冲区池中。放在默认池中的数据块，虽然可以频繁访问，但是这些段数据会老化而退出默认池，所以最好放置在保持池中，使得数据可以长久保存。
- 回收池（recycle pool）：对于随机访问的大段可以放在这个缓冲区池中，因为大的数据段会很快老化退出缓冲池，导致缓冲区的频繁刷新输出，所以需要将随机访问的大段放置在这个缓冲区池中。

在 Oracle 数据库中保持池和回收池都是用户管理的，即这两个池的大小需要手工配置，而默认池是自动管理的，在 SGA 中分配。我们通过以下指令可以查看这些保持池的大小信息，如例子 26-23 所示。

例子 26-23　查看保持池的大小信息

```
SQL> show parameter keep

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------
buffer_pool_keep                     string
control_file_record_keep_time        integer     7
db_keep_cache_size                   big integer 0
```

从输出可以看出，对于手动配置的缓冲池保持池的大小，对应的参数 db_keep_cache_size 的值为 0。

在没有设置保持池和回收池前，在数据库中只使用默认池作为数据块的缓冲池。我们可以通过例子 26-24 查询当前数据库所使用的数据库块的缓冲池。

例子 26-24　查看当前库的数据块的缓冲池

```
SQL> select id,name,block size,buffers
  2  from v$buffer pool;

     ID  NAME                  BLOCK_SIZE    BUFFERS
---------- --------------------- ------------------------ -----------------
      3  DEFAULT                 8192          47904
```

显然，当前数据库只用一个默认的数据块缓冲池，在手工设置保持池后才会显示保持池的作为数据块缓冲池的信息。

在优化时，我们需要根据实际的需求，将用户经常使用的表或者索引放在保持池 keep pool 中，接下来我们介绍如何设置保持池的大小，以及如何将数据表和索引常驻内存（保持池）中。

26.3.2　将数据常驻内存的过程

我们将用户 SCOTT 的 SALGRADE 表以及表 EMP 建立的基于函数的索引 SCOTT_EMP_INCOME_IDX 常驻保持池中。

通过上节讨论的各种数据块的缓冲池知道，保持池的大小需要手工设置，显然这个尺寸是多少应该基于常驻保持池中的数据的大小，因为我们要将一个表以及索引保存在保持池中，所以需要先确认这些数据库对象的大小，如例子 26-25 所示。

例子 26-25　查看表 SALGRADE 和索引 SCOTT_EMP_INCOME_IDX 的块大小

```
SQL> col segment name for a20
SQL>select segment name,segment type,blocks
  2  from dba_segments
  3  where owner ='SCOTT'
  4* and segment name in ('SALGRADE','SCOTT EMP INCOME IDX')

SEGMENT_NAME          SEGMENT_TYPE          BLOCKS
-------------------------------------------------------- ----------------
SALGRADE               TABLE                         8
SCOTT_EMP_INCOME_IDX  INDEX                         8
```

表 SALGRADE 和索引 SCOTT_EMP_INCOME_IDX 的大小都为 8 个数据库块大小。读者需要注意数据字典 dba_segments 是静态数据字典，如果需要获得最新的段统计信息，需要使用 ANALYZE 指令收集统计信息，如例子 26-26 所示。

例子 26-26　收集表和索引的最新统计信息

```
SQL> analyze index scott.SCOTT EMP INCOME IDX compute statistics;

索引已分析

SQL> analyze table scott.salgrade compute statistics;
```

```
表已分析。
```

那么在确认了表和索引的占用的数据块数后，数据库块大小是多少呢？我们通过例子 26-27 查询库块大小。

例子 26-27　查询的数据库块大小

```
SQL> show parameter db_block_size;

NAME                                 TYPE        VALUE
------------------------------------ ----------- --------------
db_block_size                        integer     8192
```

从输出看出当前数据库的数据块大小为 8KB。所以通过这些数据（索引和表的占用的数据块数和数据块大小）可以计算当前表和索引常驻内存需要多大，如例子 26-28 所示。

例子 26-28　计算要保存的表和索引的大小

```
SQL> select (8+8)*8 Kbytes from dual;

    KBYTES
----------
       17
```

我们需要 17KB 的保持池大小，在确认了要保存的数据的大小后，就可以手工设置保持池的大小了，如例子 26-29 所示。

例子 26-29　设置保持池的大小

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL>  alter system set db_keep_cache_size = 10MB;

系统已更改。
```

我们将保持池的大小设置为 10MB，这样可以充分满足我们要存储的包 SALGRADE 和索引 SCOTT_EMP_INCOME_IDX 大小。我们接着查看该当前数据库中数据块的缓冲池信息，如例子 26-30 所示。

例子 26-30　查询当前库的数据块的缓冲池信息

```
SQL> select id,name,block size,buffers
  2  from v$buffer_pool;

        ID NAME                 BLOCK SIZE    BUFFERS
---------- -------------------- ---------- ----------
         1 KEEP                       8192       1497
         3 DEFAULT                    8192      46407
```

从输出可以看出，多了一个缓冲池 keep，该池的数据块大小为 8KB，缓冲区大小为 1497 个数据库块大小，即 10MB。

下面我们就可以将索引和数据表设置为常驻保持池中了。在设置之前，我们先看看表 SALGRADE 和索引 SCOTT_EMP_INCOME_IDX 当前存放在什么缓冲池中，如例子 26-31 所示。

例子 26-31　查看表 SALGRADE 当前的存放在什么缓冲池中

```
SQL> connect scott/tiger@orcl
已连接。
SQL> select table_name,tablespace_name,buffer_pool
  2  from user_tables
  3  where table_name ='SALGRADE';

TABLE_NAME           TABLESPACE_NAME BUFFER_POOL
------------------------------------------------------------------------------
SALGRADE               USERS            DEFAULT
```

显示当前的表 SALGRADE 放在默认缓冲池中，因为 BUFFER_POOL 的值为 DEFAULT。下面再查看索引 SCOTT_EMP_INCOME_IDX 的缓存池。

例子 26-32　查看索引 SCOTT_EMP_INCOME_IDX 的缓存池

```
SQL> select index_name,table_name,buffer_pool
  2  from user_indexes
  3  where index_name ='SCOTT_EMP_INCOME_IDX';

INDEX_NAME                    TABLE_NAME          BUFFER_POOL
--------------------------------------------------- ------------------------
SCOTT_EMP_INCOME_IDX            EMP                 DEFAULT
```

显示当前的索引 SCOTT_EMP_INCOME_IDX 放在默认缓冲池中，因为 BUFFER_POOL 的值也为 DEFAULT。下面我们将表和索引分别设置为常驻保持池中。

例子 26-33　将表 SALGRADE 设置为常驻内存

```
SQL> alter table salgrade
  2  storage (buffer_pool keep);

表已更改。
```

现在我们通过数据字典 user_tables 查看表 salgrade 的缓冲池信息，看是否修改为常驻在保持池中，如例子 26-34 所示。

例子 26-34　查看表 SALGRADE 的缓冲池

```
SQL> select table_name,tablespace_name,buffer_pool
  2  from user_tables
  3  where table_name ='SALGRADE';

TABLE_NAME           TABLESPACE_NAME BUFFER_POOL
------------------------------------------------- --------------------------
SALGRADE              USERS              KEEP
```

从列 BUFFER_POOL 的值为 KEEP 可以知道，表 SALGRADE 已经设置为常驻内存（保持池）中了。下面我们设置索引常驻内存。

例子 26-35　将索引 SCOTT_EMP_INCOME_IDX 设置为常驻内存

```
SQL> alter index scott emp income idx
  2  storage(buffer_pool keep);

索引已更改。
```

现在使用数据字典 USER_INDEXES 查看对索引 SCOTT_EMP_INCOME_IDX 的缓冲池的修改结果，如例子 26-36 所示。

例子 26-36　查看索引 SCOTT_EMP_INCOME_IDX 的缓冲池

```
SQL> select index_name,table_name,buffer_pool
  2  from user_indexes
  3  where index name = 'SCOTT EMP INCOME IDX';

INDEX NAME                    TABLE NAME          BUFFER POOL
---------------------------------------------------- ---------------------
SCOTT_EMP_INCOME_IDX            EMP                  KEEP
```

显然，从列 BUFFER_POOL 的值为 KEEP 知道，索引 SCOTT_EMP_INCOME_IDX 已经设置为常驻内存了。

26.3.3　将常驻内存的程序恢复为默认缓冲池

我们将用户 SCOTT 的 SALGRADE 表以及表 EMP 中建立的基于函数的索引 SCOTT_EMP_INCOME_IDX 常驻保持池中。在不需要频繁访问这些表或索引时，可以将其恢复为默认缓冲池，这样就可以释放一部分内存，给其他频繁访问的数据使用。下面先演示如何将表 SALGRADE 恢复为默认缓冲池。

例子 26-37　将表 SALGRADE 恢复为默认缓冲池

```
SQL> alter table salgrade
  2  storage(buffer pool default);

表已更改。
```

接着可以查看修改结果，确认是否将表 SALGRADE 的缓冲池设置为默认缓冲池，如下例所示。

例子 26-38　查看表 SALGRADE 的缓冲池信息

```
SQL> select table_name,tablespace_name,buffer_pool
  2  from user_tables
  3  where table name ='SALGRADE';

TABLE NAME          TABLESPACE NAME BUFFER POOL
---------------------------------------------------- ---------------------
SALGRADE              USERS              DEFAULT
```

输出说明已经将表 SALGRADE 常驻内存改为使用默认缓冲区，因为 BUFFER_POOL 的值已经为 DEFAULT，以后对表 SALGRADE 的访问将把表数据读入默认缓冲区。

接下来将索引 SCOTT_EMP_INCOME_IDX 从常驻内存改为使用默认缓冲池，如下例所示。

例子 26-39　将索引 SCOTT_EMP_INCOME_IDX 恢复为默认缓冲池

```
SQL> alter index scott_emp_income_idx
  2  storage (buffer_pool default);

索引已更改。
```

其实，与设置为常驻内存不同的是，STORAGE 子句中的一个参数，将设置常驻内存的 KEEP 参数改为 DEFAULT 就修改了索引的缓冲池设置。此时，我们成功将索引 SCOTT_EMP_INCOME_IDX 的缓冲池设置为默认缓冲池。

显然此时，保持池依然占用内存，但是其中已经没有了数据，那么如何释放保持池中的内存呢？我们使用 ALTER SYSTEM 指令来回收这段内存，如下例所示。

例子 26-40　回收保持池中的内存

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL> alter system set db_keep_cache_size = 0;

系统已更改。
```

此时，我们不再使用保持池作为缓冲池，可以使用数据字典 v$buffer_pool 来验证。

例子 26-41　查看与数据库相关的缓冲池信息

```
SQL> select id,name,block_size,buffers
  2  from v$buffer_pool;

        ID  NAME                 BLOCK_SIZE    BUFFERS
----------  ---------------------------------------------  -----------------
         3  DEFAULT              8192          47904
```

显然此时只有默认缓冲池可以使用，说明保持池已经不再有效。

26.4　优化重做日志缓冲区

重做日志缓冲区是一段临时存储重做数据的内存区，用户所有变化了的数据前项和修改后的数据都保存在重做日志缓冲区中，由 LGWR 进程负责写入重做日志文件。在优化时，需要考虑该内存区的大小，以及 LGWR 的写速度和重做日志文件所在磁盘的争用等。本节我们首先重述重做日志文件的工作机制，理解和重做日志相关的等待事件，最后给你相应地解决问题的思路，达到优化重做日志缓冲区的作用。

26.4.1　深入理解重做日志缓冲区的工作机制

在 SGA 中重做日志缓冲区一般是最小的一个内存结构，在用户对数据库做更改时，重做日志缓冲区为所有修改数据的服务器进程共享使用，这些服务器进程负责将更改数据的原始值和修改后的新值以及事务 ID 写入重做日志缓冲区，而 LGWR 进程负责将重做日志缓冲区中的数据写入重做

日志文件，Oracle 的重做日志组是循环使用的，当覆盖以前的重做日志文件时，如果数据库处于归档模式则自动启动归档进程，ARCH 负责将被覆盖的重做日志文件的内容复制到归档日志文件，图 26-2 是以上行为的示意图。

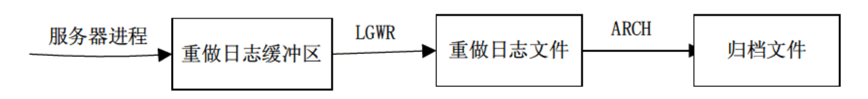


图 26-2　重做日志缓冲区以及相关进程的工作示意图

我们看到图 26-2 是一个静态图，其实在数据库内部上述活动是个十分活跃的行为，服务器进程修改数据库中的数据或表结构，不断地将相关的重做数据写入重做日志缓冲区，而重做日志缓冲区在一定的条件下，比如每 3 秒钟，将其中的重做数据写入重做日志文件，而当重做日志文件切换时（无论是用户主动切换，还是数据库自己的行为）导致归档进程 ARCH 启动，把重做日志文件中的数据读入归档文件，然后数据库才可以继续使用相关的重做日志文件，在这个过程中无论哪里出现问题都会导致一些等待事件，如果是频繁发生的等待事件，就会影响系统的性能，比如重做日志缓冲区太小，而服务器进程写入速度又比 LGWR 写出的速度快就会出现 log buffer space 等待时间，此时就需要 DBA 主动采取优化了。

为了更清楚地理解在用户修改一行数据时与重做日志相关的一系列行为，我们给出一个过程分析。

01 用户发出一条更新的 SQL 语句，该语句是某个事务的一部分，Oracle 为该事务分配了唯一的事务号。

02 服务器进程负责将需要的数据、索引和还原数据读入内存，并将要更新的行加锁。

03 服务器进程获得重做复制闩锁（闩锁实现对重做日志缓冲区的串行使用），该闩锁是服务器进程访问重做日志缓冲区的第一步。此时如果没有其他的闩锁可用，则别的服务器进程无法使用重做日志缓冲区。

04 服务器进程获得重做分配闩锁从而获得在重做日志缓冲区中的预留空间，此时释放重做分配闩锁。

05 服务器进程利用重做复制闩锁把重做项（更新数据的原始值、操作类型、事务号等信息）写入重做日志缓冲区，然后释放重做日志复制闩锁。

06 服务器进程把还原信息写入与该事务相关的还原段，还原段在用户使用 ROLLBACK 指令时使用。

07 服务器进程更新锁住的数据，将回滚所需的原始值和对数据所做的修改都写入数据库高速缓冲区。然后数据库高速缓冲区中的这些数据被标记为脏数据，因为目前内存和外存中的数据不一致。

在深入地了解了重做日志缓冲区的工作机制和过程后，我们分析 LGWR 进程何时将重做日志缓冲区的重做数据写入重做日志文件，理解这些内容对于优化重做日志缓冲区是很有必要的。以下列出的是 LGWR 把重做日志缓冲区写入重做日志文件的条件。

- 每隔 3 秒钟。
- 事务被提交时。

- 当重做日志缓冲区的记录的变化的数据量超过 1MB。
- 当重做数据的大小为重做日志缓冲区大小的 1/3 时。这里需要说明，并不是重做日志缓冲区永远不会填到超过其 1/3 容量，而是说明当重做数据量达到其容量的 1/3 这个阀值时，LGWR 进程会写出重做日志缓冲区中的数据，而剩下的 2/3 的数据可以供其他服务器进程使用。
- 检验点发生时。
- 当 DBWR 进程将数据库高速缓冲区中的数据写到数据文件前。

下面我们查看重做日志缓冲区的尺寸，如下例所示。

例子 26-42　查看重做日志缓冲区的尺寸

```
SQL> show parameter log buffer;

NAME                                         TYPE       VALUE
-------------------------------------------- ---------- ----------
log_buffer                                    integer    7024640
```

26.4.2　重做日志缓冲区相关的等待事件

如果需要优化重做日志缓冲区，必须首先确认发生了与重做日志缓冲区相关的等待事件，否则不应该随便调整重做日志缓冲区的尺寸。读者可以通过相关等待（WAIT）视图和事件（EVENT）视图，确认等待事件以及该事件涉及的文件和会话，如下所示。

例子 26-43　通过数据字典视图查看会话等待事件

```
SQL> col event for a35
SQL> col username for a10
SQL> select sw.sid,s.username,sw.event,sw.wait time
  2  from v$session s,v$session_wait sw
  3  where sw.event not like 'rdbms%'
  4  and sw.sid = s.sid
  5* order by sw.wait time,sw.event

      SID USERNAME   EVENT                               WAIT_TIME
---------- ---------- ----------------------------------- ----------
      154 SYSTEM     SQL*Net message to client                   -1
      149            Streams AQ: qmn coordinator idle wa          0
                    it

      144            Streams AQ: qmn slave idle wait             0
      147            Streams AQ: waiting for time manage          0
                    ment or cleanup tasks

      159            jobq slave wait                        0
      170            pmon timer                          0
      164            smon timer                          0

已选择 7 行。
```


虽然出现了等待事件，但是若该等待事件没有影响系统使用或系统性能就不要轻易去优化。

下面我们分析和重做日志缓冲区相关的等待事件，以及事件发生的原因分析，一旦找到等待事件并知道该事件发生的相关原因，就可以实现优化工作了。

- **Log buffer space**：该事件说明缺少重做日志的缓冲区空间，造成该等待事件的原因一般是服务器进程写入重做日志缓冲区的速度高于 LGWR 将重做日志缓冲区写出的速度，也有可能是重做日志文件所在磁盘设备速度慢或者存在设备争用，造成 LGWR 进程无法及时将重做日志缓冲区中的重做数据写入重做日志文件。

  优化方法：调整重做日志缓冲区尺寸，或者将重做日志数据文件迁移到高速磁盘上，或者为了解决争用，将重做日志文件和数据库数据文件以及归档文件放在不同的磁盘上。
- **Log file parallel write**：该事件的含义是日志文件并行写等待，是在将重做日志缓冲区中的重做数据写入磁盘引起的等待事件。造成该事件的原因一般是联机重做日志文件所在的设备速度慢或者存在磁盘争用。

  优化方法：将重做日志文件和数据库数据文件以及归档文件放在不同的磁盘上。以及将重做日志文件放置在高速盘上。
- **Log file single write**：该等待事件仅与写日志文件头块有关，表示检查点中的等待。
- **Log file switch(archiving needed)**：该等待事件的含义是日志文件切换等待，对于处于归档模式的数据库而言，当日志组写满后，在日志切换时，如果需要覆盖先前的日志，而该日志需要归档进程写入归档文件，由于写入归档文件需要时间，而 LGWR 进程需要将重做日志缓冲区中的数据写入重做日志文件，而归档未完成需要等待，在此期间就产生了 Log file switch 事件，该等待事件的原因一般是 I/O 问题、ARCH 归档进程跟不上 LGWR 日志写进程的速度或者日志组太少引起的。

  优化方法：启用多个归档 ARCH 进程或 I/O 从进程（slave process），将归档的文件和数据文件或重做日志文件放置在不同的磁盘上，减少磁盘争用以减少 ARCH 归档进程的归档事件或者增加重做日志组。
- **Log file switch(checkpoint incomplete)**：该事件是由于日志切换太频繁引起的，由于频繁地切换重做日志文件，造成检验点的排队。发生该等待事件的原因一般是重做日志缓冲区空间太小或者重做日志组太少。

  优化方法：增加重做日志组或者增加重做日志缓冲区的尺寸。
- **Log file sync**：当用户提交时，重做日志缓冲区中的数据会一次全部写到重做日志文件中，此时发生的 LGWR 的写出等待就是 log file sync 等待。造成该等待原因一般是放置联机重做日志文件的磁盘存在争用或者磁盘速度慢。

  优化方法：将重做日志文件和数据文件或归档重做日志文件放置在不同的磁盘上，以减少数据库中的各种文件之间的 I/O 争用，同时可以把重做日志文件放在高速磁盘上，以减少将重做数据写入重做日志文件的时间。
- **Latch free**：该等待事件的含义是当前的服务器进程需要某个闩锁，比如等待共享池的库高速缓存闩锁。如果发生该等待事件也可以通过数据字典 v$latch 查看相关的闩锁命中率，

如下例所示。

例子 26-44　查询与闩锁 LATCH 相关的信息

```
SQL> select latch#,name,gets,misses,1-(misses/(gets+misses)) "gets rate"
  2  from v$latch
  3  where misses>1;

    LATCH# NAME                              GETS     MISSES  gets rate
-------------------------------------- ---------- ---------- ----------
       180 dummy allocation                  12605      2      .998755445
       121 checkpoint queue latch             24231      9      .999267713
       213 shared pool                      96480     112    .998840484
     ……………….
       215 library cache lock                 45617     3      .999934239
       199 row cache objects                 80118     9      .9998872678
       214 library cache                     135162    59     .99952632677
       146 redo writing                      26541    115    .9827223526
       216 library cache pin                  7765     3     .99995883
       19 enqueue hash chains                69022     5     .9999275265
       302 session state list latch              1602     5     .99268882612

已选择 18 行。
```

可以通过以上所示的 get rate 来进一步确认闩锁的命中率。

在上述与重做日志缓冲区有关的等待事件的分析中，我们详细分析了造成这种等待事件的原因，读者或许可以体会到，理解重做日志缓冲区的工作机制，以及它涉及的各种进程对于理解这些等待事件是很有好处的，我们知道一个事件的含义，又可以分析事件发生的原因，对解决这个等待事件也就有了相应的解决思路，其实我们已经给出了解决等待事件的思路和优化方法，只是在具体操作时读者需要回顾以前章节的内容，如迁移数据文件、创建重做日志组以及向重做日志组中添加日志文件。

26.4.3　设置重做日志缓冲区大小

在发生与重做日志缓冲区相关的等待事件时，或者在 DML 操作频繁的生产数据库中，往往需要增加重日志缓冲区的尺寸。下面我们介绍如何修改重做日志缓冲区的大小并使其生效，如下例所示，先查看重做日志缓冲区的大小。

例子 26-45　查看重做日志缓冲区的大小

```
SQL> show parameter log buffer;

NAME                                 TYPE        VALUE
------------------------------------ ------------------- ------------
log_buffer                               integer     7024640
```

从输出可以看出该重做日志缓冲区的大小为 7 024 640B，所以在设置该参数的值时，也必须使用字节数据来修改该参数。为了更容易理解，我们将字节数据转换成 MB 的形式，并使用数据字典视图 v$parater，如下例所示。

例子 26-46　使用数据字典视图 v$parameter 查看重做缓冲区的大小

```
SQL> col name for a20
SQL> select name,value/(1024*1024) "MB 节"
  2  from v$parameter
  3* where name ='log buffer'

NAME                       MB
----------------------------------------- ----------
log_buffer                    6.269921875
```

我们可以看到当前数据库的重做日志缓冲区的尺寸为 7MB。由于参数 LOG_BUFFER 是静态参数，所以设置该参数后必须重新启动数据库才可以生效，下面我们增大重做日志缓冲区的尺寸，设置为 10MB。但是需要将 10MB 转换成字节（10×1024×1024=10 485 760B），如下例所示。

例子 26-47　设置重做日志缓冲区大小为 10MB

```
SQL> alter system set log buffer = 10485760 scope = spfile;

系统已更改。
```

为了值得设置的重做日志缓冲区参数生效，必须关闭数据库重新启动。下面我们关闭数据库并重新启动数据库。

例子 26-48　关闭并重启数据库

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL> shutdown immediate;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup
ORA-32004: obsolete and/or deprecated parameter(s) specified
ORACLE 例程已经启动。

Total System Global Area  734003200 bytes
Fixed Size                  171172 bytes
Variable Size             197134492 bytes
Database Buffers           52478000 bytes
Redo Buffers               11329536 bytes
数据库装载完毕。
数据库已经打开。
```

重新打开数据库后，我们修改过的重做日志缓冲区的大小生效。我们查询重做日志缓冲区的修改结果，如下例所示。

例子 26-49　查询重做日志缓冲区的尺寸

```
SQL> show parameter log_buffer;

NAME                                 TYPE        VALUE
------------------------------------ ------------------- ------------
```

```
    log_buffer                         integer     11154432
```

我们看到此时 LOG_BUFFER 的 VALUE 为 11 154 432 字节，已经不是以前的 7 024 640 字节，说明我们所做的修改生效。



我们设置的该缓冲区的大小为 10MB，但是这里的数值为 10.263726953MB，是因为 Oracle 会根据系统情况做一些调整。

26.5 优化共享池（Shared Pool）

在 Oracle 数据库系统架构中，共享池由两部分组成：库高速缓存和数据字典高速缓存。其中库高速缓存存放 SQL 语句的正文、编译后的代码以及最终的执行计划，而数据字典高速缓存存放与 SQL 语句操作相关的数据库对象，如表、索引、列以及其他对象的定义和权限信息。

对于库高速缓存而言，重用 SQL 语句可以减少硬解析的时间，从而减少 SQL 语句的响应时间。而数据字典高速缓存则减少了对 SQL 语句涉及的数据库对象和权限定义的磁盘访问，也减少了 SQL 语句的响应时间。对共享池的优化目的就是在不影响性能的条件下提高 SQL 代码以及数据字典的使用率。

26.5.1 库高速缓存

库高速缓存存放 SQL 语句的正文、编译后的代码，以及最终的执行计划。而在 SQL 语句的处理步骤中，对 SQL 语句的解析是最耗费时间的。解析需要经过 SQL 语句的语法语义分析、基于优化模式选择优化方案，以及执行最终执行计划，这种解析我们称为硬解析。而如果有相同的 SQL 语句的执行计划已经缓存在库高速缓冲区中，此时就只执行一个软分析，软分析的含义通过 SQL 语句的正文比对找到该语句的最终执行计划。所以，我们要尽量减少硬分析的发生。

26.5.2 使用绑定变量

Oracle 如何判断两个 SQL 语句是相同的呢？Oracle 对语句相同的判断条件很苛刻，要求两个语句的正文必须一样，这包括空格以及字母的大小写情况。下面所示的两个语句 Oracle 认为是不同的。

```
SQL> select ename,sal,mgr
  2  from scott.emp;
SQL> select ename,sal,MGR
  2  from scott.emp;
```

二者的区别在于 MGR 的大小写不同，虽然二者的查询结果相同，但是 Oracle 认为这两个语句是不同的，对于第二个语句会发生硬解析，大量硬解析会造成内存、CPU 以及闩的争用。

在具有如下相似的查询时，读者最好使用绑定变量，使用绑定变量 Oracle 认为使用相同条件但是不同参数值的 SQL 语句是相同的，减少硬解析的发生，如下所示。

```
SQL> select ename,sal,mgr from scott.emp where sal>5000;
```

```
SQL> select ename,sal,mgr from scott.emp where sal>2000;
SQL> select ename,sal,mgr from scott.emp where sal>3000;
```

对于上述的查询，Oraclee 认为这是 3 个独立的查询，因为 3 个语句的正文并不完全相同，所以此时，这样的查询我们最好使用绑定变量，如下所示。

例子 26-50　使用绑定变量

```
SQL> select ename,sal,mgr from scott.emp where sal>&salary
输入 salary 的值:  5000
原值    1: select ename,sal,mgr from scott.emp where sal>&salary
新值    1: select ename,sal,mgr from scott.emp where sal>5000

未选定行

SQL> select ename,sal,mgr from scott.emp where sal>&salary
输入 salary 的值:  2000
原值    1: select ename,sal,mgr from scott.emp where sal>&salary
新值    1: select ename,sal,mgr from scott.emp where sal>2000

ENAME             SAL        MGR
---------- ---------- ----------
JONES            2975       6839
BLAKE            750       7839
CLARK            2450       7839
SCOTT            3000       7566
KING             5000
FORD             3000       7566

已选择 6 行。
```

例子 26-50 通过绑定变量&salary 来代替具体的值，在查询时刻再输入具体的值，这样 Oracle 就认为使用相同的 SQL 语句，即 select ename,sal,mgr from scott.emp where sal>&salary，这样就极大地减少了硬解析的数量。

我们上面已经提过了硬分析和软分析的概念，这里再详细分析并给出一个示例说明，硬分析的过程包括 SQL 语句的语义和语法分析、检查对象和用户权限等信息，基于优化方式生成最终执行计划，最后将执行计划存在库高速缓存，显然整个过程占用大量的 CPU 时间，并会引起闩争用（因为此时的库高速缓存被占用），势必会增加 SQL 语句的响应时间。

软分析则没有 SQL 语句的语法和语义分析和基于优化方式生成执行计划的过程，比对 SQL 语句的正文（通过对语句的散列实现），发现库高速缓存中的相同 SQL 语句，以及执行该语句的执行计划。显然软分析减少了 CPU 的计算时间，又减少了闩的争用。下面我们通过启动 SQL Trace 跟踪一个 SQL 查询语句，查看是否发生硬解析。具体步骤如下所示。

01 清空共享池（目的是方便对 SQL 语句的分析）。

```
SQL> alter system flush shared pool;

系统已更改。
```

02 启动会话级的 SQL 追踪功能。

```
SQL> alter session set sql trace=true;

会话已更改。
```

03 执行 SQL 查询语句并通过 TKPROF 解释该追踪文件。

例子 26-51　执行 SQL 查询语句

```
SQL> select ename,sal,mgr from scott.emp where sal>4000;

ENAME             SAL        MGR
---------- ---------- ----------
KING             5000
```

我们开始将共享池清空，这样执行的 SQL 查询语句必须被硬解析，我们打开会话追踪文件，发现上述语句的解析过程，如下所示。

```
    PARSING IN CURSOR #3 len=50 dep=0 uid=0 oct=3 lid=0 tim=542979326426
hv=268121321
    ad='22d049b8'
    select ename,sal,mgr from scott.emp where sal>4000
    END OF STMT
    PARSE
#3:c=187500,e=304194,p=0,cr=1726,cu=1261,mis=1,r=0,dep=0,og=2,tim=54297932642
    EXEC #3:c=0,e=19,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=2,tim=5429793715
    FETCH #3:c=0,e=426,p=0,cr=7,cu=0,mis=0,r=1,dep=0,og=2,tim=5429793808
    FETCH #3:c=0,e=9,p=0,cr=1,cu=0,mis=0,r=0,dep=0,og=2,tim=5429794112
    STAT #3 id=1 cnt=1 pid=0 pos=1 obj=51151 op='TABLE ACCESS FULL EMP (cr=8 pr=0
pw=0
     time=39 us)'
```

在上述追踪文件中参数 mis 对应的值说明是否发生硬解析，如果该参数值为 1 说明发生了硬解析，因为此时的库高速缓存丢失一次命中。

下面是我们在执行一个类似的查询，看是否发生了硬解析，会使得读者对使用绑定变量有更深入的理解。

例子 26-52　执行一个与例子 26-51 类似的查询

```
SQL> select ename,sal,mgr from scott.emp where sal>3000;

ENAME             SAL        MGR
---------- ---------- ----------
KING             5000
```

此时，我们再查看追踪文件，相关内容如下所示。

```
    PARSING IN CURSOR #2 len=50 dep=0 uid=0 oct=3 lid=0 tim=26111445032
hv=269262689503
    ad='22d42677'
    select ename,sal,mgr from scott.emp where sal>3000
    END OF STMT
    PARSE
#2:c=152626,e=2264526,p=0,cr=19,cu=226,mis=1,r=0,dep=0,og=2,tim=26111445023
    EXEC #2:c=0,e=45,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=2,tim=26111445200
```

```
    FETCH #2:c=0,e=926,p=0,cr=7,cu=0,mis=0,r=1,dep=0,og=2,tim=26111445387
    FETCH #2:c=0,e=20,p=0,cr=1,cu=0,mis=0,r=0,dep=0,og=2,tim=2611144262269
    STAT #2 id=1 cnt=1 pid=0 pos=1 obj=51151 op='TABLE ACCESS FULL EMP (cr=8 pr=0
pw=0
    time=83 us)'
```

从输出可以看出此时的 mis 值为 1，说明此时使用了硬解析，因为两个 SQL 语句的正文并不相同。下面，我们再一次输入 select ename,sal,mgr from scott.emp where sal>3000 语句，继续跟踪文件，如下所示。

```
    PARSING IN CURSOR #2 len=50 dep=0 uid=0 oct=3 lid=0 tim=82689398915
hv=269262689503
    ad='22da97d8'
    select ename,sal,mgr from scott.emp where sal>3000
    END OF STMT
    PARSE #2:c=0,e=126,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=2,tim=826893989026
    EXEC #2:c=0,e=57,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=2,tim=82689399144
    FETCH #2:c=0,e=132,p=0,cr=7,cu=0,mis=0,r=1,dep=0,og=2,tim=826893993261
    FETCH #2:c=0,e=19,p=0,cr=1,cu=0,mis=0,r=0,dep=0,og=2,tim=82689399790
    STAT #2 id=1 cnt=1 pid=0 pos=1 obj=51151 op='TABLE ACCESS FULL EMP (cr=8 pr=0
pw=0
    time=119 us)'
```

从跟踪文件的输出看出，参数 mis=0 说明此时没有硬分析。

26.5.3　调整 CURSOR_SHARING 参数

该参数默认值为 EXACT，意思是只有正文完全相同的 SQL 语句才可以重用，该值也是参数 CURSOR_SHARING 的默认值，如下例所示。

例子 26-53　查看参数 CURSOR_SHARING 的默认值

```
SQL> show parameter cursor_sharing;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------
cursor_sharing                        string      EXACT
```

如果缺少绑定变量，而系统中有大量只有字面值不同的 SQL 语句，此时可以通过更改参数 CURSOR_SHARING 的值为 FORCE 来缓解 SQL 语句的硬解析开销，如果更改参数 CURSOR_SHARING 的值为 FORCE，则强制共享使用只有字面值不同的 SQL 语句。下面我们修改参数值。

例子 26-54　将参数 CURSOR_SHARING 的参数为 FORCE

```
SQL> alter system set cursor_sharing=force scope=both;

系统已更改。
```

26.5.4 设置共享池的大小

在 Oracle 10g 及以上版本中，共享池的大小由 SGA 自动调整，只要设置了 SGA_TARGET 参数，其他 SGA 组件的大小都有 Oracle 自己调整。如果是自动的 SGA 调整，我们查询共享池的大小，如下例所示。

例子 26-55 查询共享池的大小

```
SQL> show parameter shared pool size;

NAME                                 TYPE                         VALUE
------------------------------------ ---------------------------------
-----------------------------------
shared_pool_size                     big integer                  0
```

此时输出参数 shared_pool_size 的值为 0，因为此时是自动的 SGA 管理，所以 Oracle 不会显示该参数的值，在需要的时候会从 SGA 自动分配。

在安装数据库时，我们即可以选择自动 SGA 管理，此时需要告诉让 SGA 占有操作系统内存的比例或设置相应的值，如图 26-3 所示。

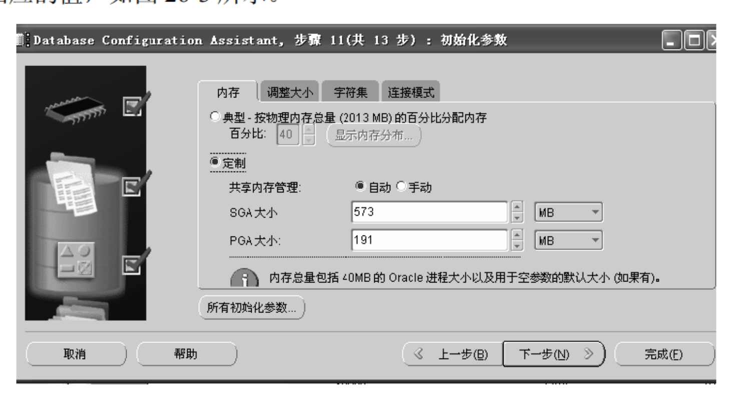


图 26-3 设置 SGA 自动管理

在安装数据库时也可以手工设置 SGA 各内存组件的大小，如图 26-4 所示。



图 26-4 手工设置 SGA 组件大小

对于手工设置 SGA 大小的方式，在数据库运行期间也可以使用 ALTER SYSTEM SET 指令修改共享池的大小，如下例所示。

例子 26-56 手工设置共享池的大小

```
SQL> alter system set shared pool size=200M scope =both;

系统已更改。
```



在 SGA 自动调整的条件下，也可以使用上述指令手工设置共享池的大小。

修改后，我们查询修改结果，验证修改是否成功，如下例所示。

例子 26-57 查询共享池大小是否修改成功

```
SQL> show parameter shared pool size;

NAME                           TYPE                      VALUE
------------------------------------------------------------------------
shared_pool_size                    big integer                    200M
```

26.6 优化数据库高速缓存（DB Cache）

数据库高速缓存中存储了最近从数据文件读入的数据块信息或用户更改后需要写回数据库的数据信息，此时这些没有提交给数据库的更改后的数据称为脏数据。当用户执行查询语句如 select * from dept 时，如果用户查询的数据块在数据库高速缓存中，Oracle 就不必从磁盘读取，而是直接从数据库高速缓存中读取，显然从内存读取的速度要快很多，这些缓存的数据由 LRU 算法管理。通过数据库高速缓存提高了用户的查询速度，减少了用户查询的响应时间。Oracle 使用 LRU 算法管理库缓冲区，把最近没被使用的数据库从库高速缓冲中删除，为其他的查询数据块保留空间。所以应该尽量设置数据库高速缓存要大些，这样就为用户提供更多的可查询缓存数据，但是 SGA 的大小是受限的，毕竟还有其他内存组件需要空间，过大的数据库高速缓存也是不可取的，本节我们讨论如何优化数据库高速缓存。

26.6.1 调整数据库缓冲区大小

数据库缓冲区存储用户读取或要修改的数据，设置合适的缓冲区大小可以提高缓冲区的命中率，提高用户数据的响应时间。如下例所示，计算数据库缓冲区命中率之前需要知道几个缓冲区值。

例子 26-58 查询相关缓冲区缓存数据

```
SQL> select name,value
  2  from v$sysstat
  3  where name in ('db block gets from cache',
  4  'consistent gets from cache',
  5  'physical reads cache');
```

```
NAME                                                              VALUE
---------------------------------------------------------------- ----------
db block gets from cache                                              47486
consistent gets from cache                                           329934
physical reads cache                                                   9813
```

我们使用数据字典视图 v$sysstat 查询了 3 个与计算数据库缓冲区命中率相关的缓冲区缓存数据，下面详细分析一下与这 3 个缓存区相关的 3 个术语：

- db block gets：DB 块获取，在数据库高速缓冲区中如果存在被更改的数据，而此数据在其他用户访问时已经提交，也就是用户访问的数据在数据库缓冲区中是最新的版本，这样的数据块读称为 db block gets，即 DB 块获取。
- consistent gets：一致获取，在数据库高速缓冲区中如果存在被更改的数据，而此数据在其他用户访问时还没有提交，也就是用户访问的数据在数据库缓冲区中是脏数据，这样的数据不会被访问，此时用户只能使用回滚段中的记录，这样的数据块读称为 consistent gets，即一致获取。
- physical reads：物理读，在数据库高速缓冲区中没有用户要访问的数据，需要从磁盘读取该数据块，这样的数据读取称为 physical reads，即物理读。

DB 块获取和一致获取都是逻辑读，在计算数据库高速缓冲区的命中率时，需要使用物理读和逻辑读的比值作为参考。即，命中率=1-物理读/逻辑读。

```
逻辑读= db block gets from cache + consistent gets from cache= 377420。
物理读= 9813。
```

所以此时数据库高速缓冲区命中率为

```
p = 1-physical reads cache/( db block gets from cache + consistent gets from cache)
= 1-9813/377420=0.973999788.
```

显然这样的库缓冲区的命中率是不错的。



命中率并不能说明此时的数据库性能就一定很好，还需要具体分析等待事件，如果此时的命中率很高，但是系统存在大量的等待事件，如数据文件离散度等，可能存在大量的全表扫描，就需要进一步分析等待事件和相关的 SQL 语句。

如果对自己的应用环境非常熟悉，可以采用手工设置数据库高速缓冲区的尺寸，此时在数据库系统的典型运行阶段查询命中率，反复调整直到命中率满足要求。如下例所示，修改库高速缓冲区大小。

例子 26-59 修改库高速缓冲区大小

```
SQL> alter system set db cache size=192M;

系统已更改。
```

26.6.2　使用缓冲池

如果用户访问使用了多个表，并且这些表都相当大，此时这些表只有部分保存在库高速缓冲区中，库缓冲区中的大表数据的存在会降低数据库缓冲区的命中率，而如果这样的大表不是用户频繁访问的对象，就是可以使用回收缓冲池（recycle pool）来存放，这样的数据库对象会被覆盖。

如果用户访问对多个小表全表扫描，且这样的行为频繁发生，就可以将这些小表保存在保留池（keep pool）中，这样的数据库对象不会被覆盖掉。

对于其他既没有存储在回收缓冲池，也没有存储在保留池中的数据库对象，默认存储在数据库缓冲区的默认池（default pool）中。



缓存表的尺寸不要超过缓冲区大小的 10%。

下面是将数据对象存储在缓冲池中的语法。

```
SQL> create index t_idx Storage (buffer_pool keep);
SQL> alter table t name storage(buffer pool recycle);
SQL> alter index t_idx  storage(buffer_pool keep);
```

下面，我们给出一个例子说明如何创建一个索引的同时将该索引调入保留池。

例子 26-60　创建索引并将其缓存在保留池

```
SQL> create index scott emp ename
  2  on scott.emp(ename)
  3  storage (buffer_pool keep);

索引已创建。
```

为了验证保存结果，我们通过数据字典 USER_INDEXES 视图查看 BUFFER_POOL 列的值，如果该值为 KEEP 则说明索引已经保存在保留池中。

例子 26-61　查询索引 SCOTT_EMP_ENAME 是否缓存在保留池中

```
SQL> col buffer pool for a15
SQL> run
  1  select index_name,table_name,buffer_pool
  2  from user indexes
  3* where index name='SCOTT EMP ENAME'

INDEX_NAME           TABLE_NAME                BUFFER_POOL
----------------------------------------
--------------------------------------------------- ------------------------
SCOTT_EMP_ENAME              EMP                          KEEP
```

从输出看出，索引 SCOTT_EMP_ENAME 已经缓存在保留池中，因为列 BUFFER_POOL 的值为 KEEP。

如果是表，则可以通过数据字典 USER_TABLES 来查看，该视图的 CACHE 列说明该表是否被缓存，BUFFER_POOL 列说明该表缓存的缓冲池名称。下面我们将表 EMP 放入回收池。如下所示。

例子 26-62　将表 EMP 缓存在回收池

```
SQL> alter table scott.emp
  2  storage (buffer_pool recycle);

表已更改。
```

此时，我们将 SCOTT 用户的 EMP 表存储在回收池中。下面通过数据字典 DBA_TABLES 查看更改结果，如下例所示。

例子 26-63　查看表 EMP 的缓存信息

```
SQL> select table_name,cache,buffer_pool
  2  from dba_tables
  3  where table name='EMP';

TABLE NAME                    CACHE      BUFFER POOL
-------------------------------------------------- --------------------------
EMP                             Y         RECYCLE
```

列 CACHE 说明表 EMP 已经缓存在缓冲区中，而列 BUFFER_POOL 说明对应的缓冲区为回收池 RECYCLE POOL。

我们也可以使用 CACHE 关键字，在创建一个表时，说明如果该表被缓存在默认池中，如下例所示。

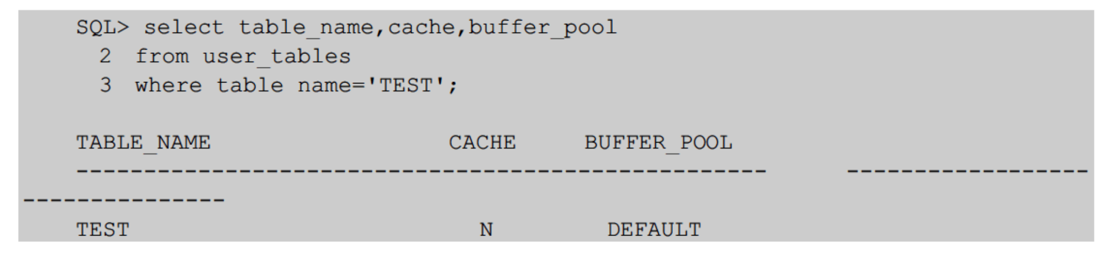
例子 26-64　使用 CACHE 关键字将表缓存在默认池中

```
SQL> create table test
  2  as
  3  select *
  4  from scott.dept
  5  cache;

表已创建。
```

下面，通过数据字典 USER_TABLES 查看表 TEST 的缓存结果，如下例所示。

例子 26-65　查看表 TEST 的缓存信息



```
SQL> select table_name,cache,buffer_pool
  2  from user_tables
  3  where table name='TEST';

TABLE_NAME                    CACHE      BUFFER_POOL
-----------------------------------------------------      ------------------
---------------
TEST                            N          DEFAULT
```

此时，我们发现表 TEST 会使用 DEFAULT 缓冲池，但是目前没有被缓存，所以，还需要使用如下指令启动表的缓存。

例子 26-66　将表 TEST 缓存在默认池中

```
SQL> alter table test cache ;
```

```
表已更改。
```

我们继续通过数据字典 USER_TABLES 查看表 TEST 的缓存结果，如下例所示。

例子 26-67　查看表 TEST 的缓存信息

```
SQL> select table_name,cache,buffer_pool
  2  from user_tables
  3  where table_name='TEST';

TABLE_NAME                   CACHE      BUFFER_POOL
-------------------------------------- ------------------ ---------------
TEST                            Y          DEFAULT
```

此时的列 CACHE 的值为 Y，说明已经将该表加载到缓冲区，此缓冲区为 DEFAULT 缓冲区。

26.7　优化PGA内存

PGA 是程序全局区，该区域在数据库会话专有连接模式下是私有区域，服务器进程和用户进程一一对应，用户进程单独使用 PGA，在共享服务器会话连接模式下，一个服务器进程对应多个用户进程，PGA 是多个用户进程共享使用。PGA 主要用作大规模的数据排序，如用户输入的 SQL 语句有 GROUP BY 或 ORDER BY 子句等操作。

所谓的 PGA 优化就是将这些大规模的数据排序放在 PGA 中运行，而不是使用虚拟内存而占用操作系统的交换区（SWAP AREA）。这样就要求合理地设置 PGA 的大小，从而使得排序区符合系统需要，在 Oracle 9i 之前，该排序区由参数 SORT_AREA_SIZE 决定，通过手工设置该参数，反复调整以满足系统需要。可以通过如下的方式查看该参数的值。

例子 26-68　查看 PGA 中排序区的大小

```
NAME                                TYPE       VALUE
--------------------------------------------------------------- ----------
sort_area_size                            integer        65536
```

如果处于大规模排序的需要，可以调整该参数为更大的值，但是问题是到底多少是最好的。显然如果分配过多，则造成系统的其他组件的内存不足，如果过少又不能满足排序的需要。在 Oracle 9i 及以上版本（Oracle 10g、Oracle 11g）中支持 PGA 排序区的自动调整功能。但是该自动调整有一个限制就是必须给出一个排序区的值，排序区会根据排序需要而在这个值内自动调整。该值由参数 PGA_AGGREGATE_TARGET 决定，同时为了启动 PGA 排序区的自动管理必须设置参数 WORKAREA_SIZE_POLICY 为 AUTO，这也说明要设置 PGA 排序区的自动管理必须配置两个参数，即 PGA_AGGREGATE_TARGET 和 WORKAREA_SIZE_POLICY。下面我们查看系统上这两个参数的值。

例子 26-69　查看 PGA 的排序区是否为自动管理

```
SQL> col name for a20
SQL> col value for a30
SQL> select name,value,isdefault
```

```
  2  from v$parameter
  3* where name in ('pga_aggregate_target','workarea_size_policy')

NAME                  VALUE                                      ISDEFAULT
-------------------------------------------------------- -------------------------
pga_aggregate_target     200278016                                 FALSE
workarea_size_policy     AUTO                                   TRUE
```

从例 26-69 的输出可以看出参数 WORKAREA_SIZE_POLICY 是默认的参数，因为其 ISDEFAULT 的值为 TRUE，而参数 PGA_AGGREGATE_TARGET 是需要设置的，它不是系统的默认参数，因为 ISDEFAULT 的值为 FALSE。

下面我们分析一下当前数据库的 PGA 状态，如下例所示。

例子 26-70　通过数据字典视图 v$pgastat 查询 PGA 状态信息

```
SQL> col name for a40
SQL> run
  1  select *
  2* from v$pgastat

NAME                                    VALUE      UNIT
----------------------------------------------------- ------------
aggregate PGA target parameter            200278016     bytes
aggregate PGA auto target               16797888     bytes
global memory bound                     40054784     bytes
total PGA inuse                        19604480     bytes
total PGA allocated                     42294272      bytes
maximum PGA allocated                   90947584     bytes
total freeable PGA memory                      0      bytes
process count                                21
max processes count                           33
PGA memory freed back to OS                    0    bytes
total PGA used for auto workareas                0       bytes

NAME                                              VALUE UNIT
--------------------------------------------- ---------- ------------
maximum PGA used for auto workareas         3831808   bytes
total PGA used for manual workareas               0    bytes
maximum PGA used for manual workareas              0  bytes
over allocation count                            0
bytes processed                        101315584  bytes
extra bytes read/written                          0    bytes
cache hit percentage                          100  percent
recompute count (total)                       869

已选择 19 行。
```

需要注意，以上输出中记录 PGA 状态的 3 个参数，即 aggregate PGA target parameter、aggregate PGA auto target 和 cache hit percentage。其含义如下所示。

- aggregate PGA target parameter：用户设置的当前 PGA 的内存总和。
- aggregate PGA auto target：Oracle 为 PGA 的排序区分配的内存大小，显然其值不能超过

PGA 内存的总和。

- cache hit percentage：说明排序区在 PGA 的排序区完成的比例，100 percent 表示全部排序都在 PGA 的排序区内完成，所以 Oracle 自动分配的排序区的尺寸是合理的。

我们可以使用以下方式监控 PGA 的排序区是否合理，其目的是观察参数 cache hit percentage 的值，如果是 100 percent 则认为 Oracle 根据系统状态设置的 PGA 的排序区是合理，否则就需要增加参数 PGA_AGGREGATE_TARGET 的值，以增加为 PGA 的排序区添加内存的需要。如下例所示，监控 PGA 的排序区。

例子 26-71　查看在 PGA 的排序区进行排序的百分比

```
SQL> select *
  2  from v$pgastat
  3  where name like 'cache%';

NAME                                            VALUE  UNIT
----------------------------------------------------- ---------- ---------
cache hit percentage                                      100     percent
```

显然其中的 VALUE 值为 100，所以当前的排序行为都在 PGA 的排序区内完成。如果出现部分排序不在 PGA 的排序区完成，即 VALUE 的值小于 100，则需要考虑适当增加 PGA 的内存总和。如下例所示，通过设置参数 PGA_AGGREGATE_TARGET 来调整 PGA 的内存总和。

例子 26-72　调整 PGA 的内存大小

```
SQL> alter system set pga aggregate target = 76M;

系统已更改。
```

我们再查看修改结果，如下所示。

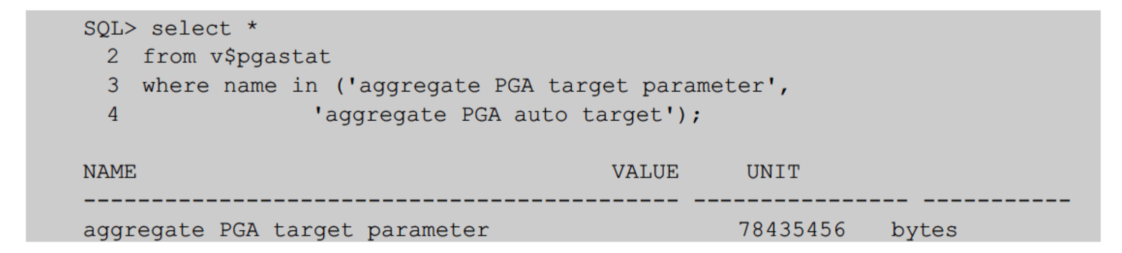
例子 26-73　查看参数 PGA_AGGREGATE_TARGET 的修改结果

```
SQL> show parameter pga

NAME                               TYPE        VALUE
---------------------------------------- ------------------- ------------
pga_aggregate_target                      big integer     76M
```

我们再次查看此时 PGA 的状态信息，看 Oracle 是为 PGA 的排序区分配了多少内存，如下例所示。

例子 26-74　查看 PGA 的大小以及 PGA 中排序区的大小



```
SQL> select *
  2  from v$pgastat
  3  where name in ('aggregate PGA target parameter',
  4              'aggregate PGA auto target');

NAME                                           VALUE      UNIT
-------------------------------------------- ---------------- -----------
aggregate PGA target parameter                      78435456    bytes
```

```
aggregate PGA auto target                        22379217    bytes
```

我们看到此时的 PGA 的总和为 76MB，同时自动分配给 PGA 的排序区的尺寸为 213.424805MB，可见随着 PGA 的总内存的增加 Oracle 会自动增加其为排序区的分配的内存容量。

读者也可以通过数据字典视图 v$pga_target_advice 获得 PGA 的排序区进行排序行为更详细的信息。

例子 26-75　查询 PGA 的排序区排序的详细信息

```
SQL> select pga target for estimate/(1024*1024) as estd target,
  2  estd pga cache hit percentage ,estd overalloc count
  3  from v$pga target advice;

ESTD_TARGET ESTD_PGA_CACHE_HIT_PERCENTAGE ESTD_OVERALLOC_COUNT
--------------------------------------------------------------------------
        32                              100                  0
        64                              100                  0
       17                              100                  0
       192                              100                  0
       76                              100                  0
 307.199219                              100                  0
 358.399414                              100                  0
 409.599609                              100                  0
 460.799805                               100                  0
       512                              100                  0
       768                              100                  0

ESTD TARGET ESTD PGA CACHE HIT PERCENTAGE ESTD OVERALLOC COUNT
--------------------------------------------------------------------------
      1024                              100                  0
      1536                              100                  0
      2048                              100                  0

已选择 14 行。
```

从以上输出看出，Oracle 给出了一系列的对 PGA 的估计值，并且给出了每个估计值状态下排序在 PGA 的排序区完成的比率，由参数 ESTD_PGA_CACHE_HIT_PERCENTAGE 决定。由于笔者的计算机上没有任何排序行为，所以对于 PGA 的每一个估计值所有的排序都在 PGA 的内存中完成。如果在生产数据库系统上，参数 ESTD_PGA_CACHE_HIT_PERCENTAGE 的值一般不会都是 100，这样可以根据这个参数的提示，来确定手动设置的 PGA 内存总和是否合适，合适的含义是要求所有的排序行为都在 PGA 的内存中完成，这样通过内存中排序减少了由于排序内存不足而造成的 I/O 开销。

26.8　本章小结

Oracle 实例是一个非常重要的概念，它包含一组 Oracle 数据库相关的内存组件和一些后台进程。在实例优化中，我们先介绍了将程序或者数据常驻内存，这对于用户经常使用的程序如存储过

程或函数、用户频繁访问的表等是十分有效的。

Oracle 实例的内存组件包括数据库高速缓存、共享池、重做日志缓冲区、PGA，以及大池和 Java 池。本章我们讨论了共享池的优化、重做日志缓冲区的优化、数据库高速缓存的优化，以及 PGA 的优化。无论是哪个组件的优化，我们都首先需要理解这些组件的作用，然后学会如何调整这些组件的参数，在介绍如何优化时，我们也按照这样的逻辑来设计，希望读者学习时要时时回忆该组件的作用，这对于学习优化是十分重要的。